IBM® Personal Systems Developer

A Publication of the IBM Developer Assistance Program
Spring 1991

- Client-Server Programming
- Application Enablers
- Spotlight on SAS

IBM Personal Systems Developer

Table of Contents

The *IBM Personal Systems Developer* is published quarterly by IBM Software Developer Support, Internal Zip 2212, 1000 N W 51 Street, Boca Raton, Florida 33429. Phone (407) 982-6408, Fax (407) 443-4233. IBM employees and M&S branch office customers can subscribe to the Personal Systems Developer through IBM Mechanicsburg's Systems Library Subscription Service (SLSS) using the Developer's order number, G362-0001. Others can subscribe by calling the publisher, Graphics Plus Inc., directly at (800) READ-OS2. Subscriptions (U.S. only) are $39.95 yearly. Questions, suggestions and article ideas should be sent to the Editor.

While back issues are not generally available, articles from the first seven issues of the Personal Systems Developer have been published in *OS/2 Notebook: The Best of the IBM Personal Systems Developer.* This book can be bought at a local bookstore ($29.95) or by calling Microsoft Press at (800) MS-PRESS. Its Mechanicsburg order number is G362-0003-00.

Editor: Dick Conklin, IBM Software Developer Support. Publisher: Graphics Plus, Inc., 640 Knowlton Street, Bridgeport, CT 06608,
Project Manager: Jo-Ann Radin.

IBM Personal Systems Developer is published by the Entry Systems Division of International Business Machines Corporation, Boca Raton, Florida, U.S.A., Dick Conklin, Editor.

Titles and abstracts, but no other portions, of information in this publication may be copied and distributed by computer-based and other information service systems. Permission to republish information from this publication in any other publication or computer-based information system must be obtained from the Editor.

IBM believes the statements contained herein are accurate as of the date of publication of this document. **However, IBM hereby disclaims all warranties either expressed or implied, including without limitation any implied warranty of merchantability or fitness for a particular purpose. In no event will IBM be liable to you for any damages, including any lost profits, lost savings or other incidental or consequential damage arising out of the use or inability to use any information provided through this publication even if IBM has been advised of the possibility of such damages, or for any claim by any other party.**

Some states do not allow the limitation or exclusion of liability for incidental or consequential damages so the above limitaion or exclusion may not apply to you.

This publication may contain technical inaccuracies or typographical errors. Also, illustrations contained here may show prototype equipment. Your system configuration may differ slightly.

This publication may contain articles by non-IBM authors. These articles represent the views of their authors. IBM does not endorse any non-IBM products that may be mentioned. Questions should be directed to the authors.

This information is not intended to be an assertion of future action. IBM expressly reserves the right to change or withdraw current products that may or may not have the same characteristics or codes listed in this publication. Should IBM modify its products in a way that may affect the information contained in this publication, IBM assumes no obligation whatever to inform any user of the modification.

It is possible that this material may contain reference to, or information about, IBM products (machines and programs), programming or services that are not announced in your country. Such references or information must be construed to mean that IBM intends to announce such products, programming or services in your country.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever.

All specifications are subject to change without notice.

To correspond with the *IBM Personal Systems Developer*, please write to the Editor at IBM Corporation, Internal Zip 4607, P.O. Box 1328, Boca Raton, FL 33429-1328.

Trademarks

IBM, PS/2, Operating System/2, OS/2, DisplayWrite, Audio Visual Connection, and AVC are registered trademarks of IBM Corporation.

Communication Manager, Database Manager, Presentation Manager, OS/2 Application Manager, Systems Application Architecture, SAA, Application System (AS), IBMLink, CICS, NetView, LAN Requester, and NetBIOS are trademarks of IBM Corporation.

Microsoft, MS, MS-DOS, Excel, CodeView, WORD, and Windows are registered trademarks of Microsoft Corporation.

Apple, Mac and Macintosh are registered trademarks of Apple Computer, Inc.

DeScribe is a trademark of DeScribe, Inc.

SUN is a registered trademark of Sun Microsystems, Inc.

UNIX is a registered trademark of UNIX System Laboratories, Inc.

InfoAlliance is a trademarked datasource integration product from Software Publishing Corporation.

Lotus, Freelance and 1-2-3/G are registered trademarks of Lotus Development Corporation.

DEC and DECnet are registered trademarks of Digital Equipment Corporation.

SAS, SAS Institute, SAS Applications System, SYSTEM 2000, SAS/C, JMP Software and SAS/ASSIST are registered trademarks of SAS Institute Inc.

MultiVendor Architecture is a trademark of SAS Institute Inc.

Novell, Novell Netware Requester and Netware SQL are registered trademarks of Novell, Inc.

EASEL is a registered trademark of Interactive Images, Inc.

AutoCAD is a trademark of Autodesk, Inc.

PIPES Platform is a trademark of PeerLogic.

Ashton-Tate, Framework and dBase are registered trademarks of Ashton-Tate Corporation.

Relish is a trademark of Sundial Systems Corporation.

WideAngle is a trademark of Inner Media, Inc.

SCO is a registered trademark of Santa Cruz Operation, Inc.

Editor's Comments

Dick Conklin

This is another of our *theme* issues on a topic that fits OS/2 like a glove: Client-Server Programming. Our lead article was written by IBMers Bob Orfali and Dan Harkey, based on their new book. Other contributors examine client-server programming from various angles: cooperative processing, databases, DDE, APPC, CPI-C, even using Client-Server architecture within a standalone application.

The new IBM-Novell® agreement shows that IBM is serious about making OS/2 an open platform for client-server solutions. An OS/2 machine will now be able to act as a server for Macs®, UNIX® machines, and thousands of DOS machines that run Netware.

In addition, we're starting a new section in this issue, Application Enablers, with articles about three OS/2 products: CICS™ for OS/2, InfoAlliance™, and the PIPES Platform™. Our Spotlight feature article is on a leading-edge software company that wrote the book on application portability: SAS.

In our next issue we'll examine a popular topic: a comparison of the Windows 3.0™ and OS/2 PM application platforms. We'll look at these two popular graphical interfaces with special attention to multiple platform development and application migration.

We'll also have an article by Micrografx, a company with a lot of OS/2 and Windows development experience. In April, Micrografx and IBM announced a development agreement for OS/2 2.0 migration tools and performance enhancements. (At the same meeting, IBM announced new pricing for OS/2 1.3 versions: $150 (U.S.) for Standard Edition and $690 for Extended Edition. DOS-to-OS/2 upgrades are $99 for Standard and $635 for Extended Editions.)

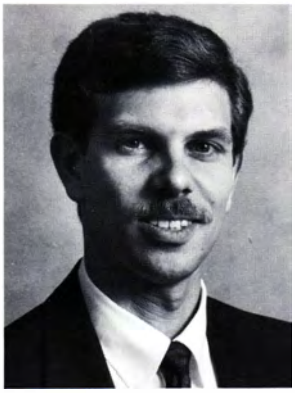
You won't want to miss any of our future issues. So, if you are still reading borrowed copies, it's not too late to subscribe. Just call our subscription hotline, (800) READ-OS2. If you are missing some back issues, the best way to catch up is by getting a copy of *OS/2 Notebook: The Best of the IBM Personal Systems Developer*. It's in your local bookstore or you can call (800) MS-PRESS to order one.

Dick Conklin

Dick Conklin, Editor

Developer Assistance Program

Developer Assistance Program Update



Joe Carusillo

by Joe Carusillo

I would like to take this opportunity to introduce myself. I am the Manager of Software Developer Programs for the IBM Personal Systems Line of Business. My responsibilities include both the Developer Assistance Program (DAP) and the OS/2 Conversion Workshop Program. This is a new assignment for me. Prior to this, I had been managing the Conversion Workshop Program exclusively. Those of you who read this column regularly know that these workshops are a special service of the DAP. They are a very intensive, one week, hands-on program aimed at helping software developers convert their DOS and Windows applications to fully support OS/2. To date we have had over 150 software companies participate in one or more of the three different workshops currently offered. A number of these new OS/2 products are already available. The remainder — for the most part — are scheduled to be released by year-end.

Since joining IBM in 1982, I have had the pleasure of working with a large number of software developers. These efforts have included the development of some of IBM's own PC applications and helping out developers with their own product development. I am looking foward to the opportunity the DAP offers to greatly expand my working relationships with the software developer community here in the U.S. If we have not yet met, perhaps we will soon at one of our workshops, seminars, a PS/2 Forum, or maybe at a business show later in the year.

It's still a little early for me to be very specific on what my personal agenda for the DAP will be. I've already had many suggestions from both inside and outside of IBM on what the priorities for 1991 should be. The types of things that are currently on the table are: expansion of the DAP worldwide, offering an IBM version of the OS/2 Version 2.0 SDK direct from the DAP, providing a lab with a wide range of PS/2 hardware that developers can schedule time in for product testing, adding selected IBM Micro Channel adapters to the existing discount program, and doing more to raise the visibility of the DAP to the industry at large. Believe it or not, there are still some developers that don't know about our services or how to contact us directly. I'll keep you informed about these things in future issues. Check the requirements below to see if you qualify for the DAP.

ELIGIBILITY AND ENROLLMENT

The IBM Developer Assistance Program is for software developers working on products for commercial release. You are eligible to participate in the program if:

- You develop products that support IBM OS/2 or IBM DOS.
- Your company is a U.S. company or the U.S. subsidiary of a non-U.S. company.
- You are currently marketing your products.

If you are not currently marketing a product, you can submit a non-confidential, detailed business plan showing marketing and development activities and schedules. IBM reserves the right to accept or reject an application based on this business plan. There is no charge for enrollment in the Developer Assistance Program.

To request an application form or ask questions about the program, contact Developer Support:

IBM Corporation-Developer Support
Internal Zip 4607
1000 NW 51 Street
Boca Raton, FL 33429-1328
(407) 982-6408

OS/2 CONVERSION WORKSHOPS

The IBM Developer Assistance Program currently offers three OS/2 Conversion Workshops:

Workshop 1	Converting a DOS Application to OS/2
Workshop 2	Developing a Presentation Manager™ (PM) interface for an OS/2 VIO Application
Workshop 3	Converting a Windows 3.0 Application to OS/2 Presentation Manager

Workshop Format: The workshops combine lectures and lab time. Lectures, in a classroom setting, focus on the specific workshop topic. A laboratory next to the classroom contains workstations for attendee use during workshop lab sessions. Attendees will work on their own applications using the information learned in the classroom lectures. The lab remains open each evening until 10:00 PM (technical assistance is available until 6:00 PM).

Attendance: By invitation only. For qualified companies. Up to two attendees per company, working as a team, may participate. Seating is normally limited to twelve programmers per session.

Financial Considerations: Each workshop, including continental breakfast and lunch, is free of charge. Attendees are responsible for all other expenses (hotel, travel, dinner, incidentals, and so on).

For additional information about this extraordinary opportunity from IBM, please contact Jean Hicks (407) 982-0165 or Marilyn Johnson (407) 982-5514.

1991 IBM OS/2 TECHNICAL SEMINARS

The OS/2 Technical Seminar is an excellent chance to get updated on what's new with OS/2—particularly version 2.0. These week-long seminars are designed for experienced OS/2 software developers, both vendor and corporate customers. They include main tent and elective sessions with demonstrations and time for hands-on. Topics include OS/2 and Windows Considerations, Developer Tools and Toolkits, OS/2 Extended Edition, Performance and Tuning, Multimedia, the 32-Bit API, Presentation Manager and LAN Server enhancements, and other topics.

The $1495 (U.S.) seminar fee includes all class sessions, meals include breakfast and lunch daily, the seminar workbook, OS/2 version 2.0 prototype code, sample programs, tools and documentation. Seminars are planned for Dallas (May), Toronto (June), Washington DC (July), Southern California (September), Australia (August) and Europe (September).

If you would like information about these seminars (schedule, enrollment deadlines) or for U.S. enrollments, please call (800) 548-2464, Monday through Friday, 8:00-4:40 Central time. In Canada call (800) 465-1234.

Joe Carusillo, *IBM Personal Systems Programming, 1000 NW 51st Street, Boca Raton, FL 33429. Mr. Carusillo is the Manager of Software Developer Programs which includes the Developer Assistance Program. He started with IBM Entry Systems in 1982 and has worked on PC software since that time. He has a BS in Computer Systems Engineering from the Columbia University, School of Engineering and Applied Science, in New York City.*

Developer Assistance Program

Software Maintenance Update

Jeffry Ullman

by Jeffry Ullman

This is an update to the "Software Maintenance" articles that appeared in the IBM Personal System Developer Winter 1990 issue (pages 12-18) and the Summer 1990 issue (pages 7-9). Please reference these articles for a detailed description of the underlying concepts and terminology.

OS/2 EXTENDED EDITION V1.2/1.3 REINST COMMAND

OS/2 Extended Edition V1.2/1.3 uses the REINST command to selectively install the Communications Manager, Database Manager, and/or LAN Requester. The REINST command is also used to add features to an already installed system. OS/2 CSDs only update code that is physically resident on a hard file. Therefore, the current OS/2 CSD should be reapplied after each use of the REINST command.

Example: A user has an OS/2 system with the base operating system and the Communications Manager installed. This user has always applied the most current CSD to maintain his system. The user then decides to use the REINST command to install the Database Manager. Since the Database Manager was never physically resident on the the system, the OS/2 CSD process could not update any Database Manager code. To update the newly installed Database Manager, this user should reapply the most recently applied OS/2 CSD. This same situation will occur whenever any Communication Manager configuration (CFG) file is updated and the REINST command is used to install new code based on the updated configuration file.

AUDIO-VISUAL CONNECTION (AVC)

The current release of the Audio-Visual Connection® program for OS/2 is V1.03. IBM MultiMedia Solutions has also released maintenance for V1.03. However, this has not changed the version number of the program. Both V1.03 and the maintenance to V1.03 can be obtained at no charge by calling IBM MultiMedia Solutions at (800) 627-0920. They are available to any original licensed owner of the software regardless of where the software was acquired (authorized dealer, industry remarketer, or IBM branch office).

EASEL V1.1 FOR OS/2 EXTENDED EDITION

IBM has released several program enhancements to EASEL V1.1 for OS/2 Extended Edition. The enhancements include:

- Support for Dynamic Data Exchange (DDE)
- 3270/5250 Merged Verb Set Support
- Permanent and Temporary Dialog Support
- Double Byte Character Set (DBCS) Support

The documentation for the program enhancements are:

Developer's Guide Supplement-Dynamic Data Exchange Support (IBM form # SC38-7510)

Developer's Guide Supplement-DBCS, 3270/5250, and Dialogs (IBM form # SC38-7511)

Please note that this documentation can only be ordered from an IBM branch office by form number. Part numbers are not applicable.

Program enhancements can only be obtained by requesting the current PTF for EASEL V1.1 for OS/2 Extended Edition. The PTF may be requested by calling the IBM Support Center at (800) 237-5511.

DISTRIBUTED CONSOLE ACCESS FACILITY (DCAF)

IBM has announced a PTF for the Token Ring 16/4 Trace and Performance (TAP) Program. This PTF will allow Distributed Console Access Facility (DCAF) to control and/or monitor a remote TAP workstation. No PTF to DCAF is required for this DCAF enhancement. The PTF for the TAP Program is listed in Figure 1 and is available from the IBM Support Center at no charge.

PROGRAM PRODUCT	PTF	CSD
OS/2 Standard Edition V1.2	XR00429	XR04095
OS/2 Extended Edition V1.2	WR00557	WR04098
OS/2 LAN Server V1.2	WR00557	WR04098
Token Ring Network 16/4 Trace and Performance Program	UR33813	
OS/2 Programming Tools and Information V1.2	XR00441	XR04096
EASEL V1.1 for OS/2 Extended Edition	UR32887	
Audio Visual Connection		V1.03
Realtime Interface Coprocessor OS/2 Support	UL79292(V1.03)	
Realtime Interface Coprocessor DOS Support	UL76230(V1.02)	
Realtime Interface Coprocessor C Language Support	UL79455(V1.02)	
NetView/PC V1.2.1 (OS/2)	UR32747	
NetView/PC V1.1 (DOS)	UR24810	
PC Node Manager V1.1	UB01244	
DOS 4.0	UR31300	
LAN Manager V2.0	UR27865(V2.01)	
Personal Communications/3270 V1.01	IP00625	2010
3270 Entry Level V1.21	UR29500(V1.22)	
3270 Workstation Program V1.10	UR23217(V1.12)	
Current V1.1		Corrective Service

Figure 1. Partial List of Available PS/2 Maintenance (Continued)

PROGRAM PRODUCT	PTF	CSD
DisplayWrite 5		Modification Level 02.1
DisplayWrite 5 Composer		Modification Level 02.1
DisplayWrite 5/2		Modification Level 03
DisplayWrite 5/2 Composer		Modification Level 03
"AS IS" SOFTWARE	**IBM LINK ITEM/IBM PUBLICATION**	
COBOL/2	8VLBV(V1.02)	
FORTRAN/2	277MP(V1.11)	
Pascal/2	719MP	
8514/A Adapter Interface Driver V1.02	Form Number S15F-221 Part Number 15F2215	

Figure 1. Partial List of Available PS/2 Maintenance

IBM PC BOOKS & SPECIAL PROMOTION

IBM PC Books & Special Promotions is a fulfillment house that IBM uses to distribute PS/2 related items. They can be reached at (800)-426-7282. Items that can be requested at no charge are:

ITEM	PART NUMBER
• DASDDRVR.SYS V1.56	64F1500
• PS/2 Model 90 XP 486 Reference Diskette V1.02	57F1952
• PS/2 Model 95 XP 486 Reference Diskette V1.02	57F1959
• OS/2 Standard Edition 1.3 L40SX Refresh Level 1.30.1	92F2693
• OS/2 Extended Edition 1.3 L40SX Refresh Level 1.30.1	90X7975
• OS/2 Extended Edition V1.3 Problem Determination Guide for the Service Coordinator	01F0301

Version 1.02 (or higher) reference diskettes are required on PS/2 models 90 & 95 with more than 8MB of memory.

PS/2 SPECIAL PROMOTIONS

IBM has determined that with certain PS/2 models and under certain circumstances data can be lost with the 60MB hard disk and the 160MB Small Computer System Interface (SCSI) hard disk. IBM has released software utilities that will correct the problem. PS/2 Special Promotions is a fulfillment house that IBM is using to distribute these no-charge utilities. They can be reached at (800) 845-4263. The following utilities can be requested:

ITEM	PART NUMBER
60MB Fixed Disk Buffer Test Update Diskette	91F8648
160MB SCSI Hard Disk Mode-Select Restoration Diskette	91F8579

DISPLAYWRITE 5

The IBM Support Center now provides defect and usage support for DisplayWrite 5 and DisplayWrite 5 Composer to any original licensed owner regardless of where the software was acquired (authorized dealer, industry remarketer, or IBM branch office.)

Figure 1 is a partial list of available PS/2 maintenance. All program product maintenance is available via the IBM Support Center, IBMLink, ServiceLine, or MultiMedia Solutions Help Line. IBMLink items are contained in the Technical Q & A database. IBM publications can be ordered from an authorized dealer or IBM branch office.

Jeffry Ullman, *IBM United States, South Florida Trading Area, Branch Office JHZ, Columbus Center, One Alhambra Plaza, Coral Gables, FL 33134. Mr. Ullman is a Workstation Specialist and Account Systems Engineer on the Florida Power and Light account in Miami, FL. He specializes in OS/2, the DOS 3270 emulation family and S/370 programmable and non-programmable workstation connectivity. He joined IBM in Jacksonville, Florida in 1983. Mr. Ullman has a Bachelor of Business Administration in Marketing and Master of Business Information Systems from Georgia State University.*

Spotlight

The SAS® Applications System for OS/2

by Peter D. Goldstein

SAS Institute Inc.

The SAS® Applications System for OS/2 is a real paradox in the OS/2 software market. It is one of the few products that simultaneously exploits all of OS/2's powerful features and maintains strict interoperability with versions of the same product running on nearly every mainframe, mini-, and microcomputer platform on the market today. With their OS/2 product, SAS Institute has dispelled any notions that sacrifices have to be made to accommodate both needs. In this issue's Spotlight, we give you a look into the development operations of SAS Institute, insight into their programming philosophies, and details on how they have taken full advantage of OS/2 and Presentation Manager.

SAS INSTITUTE INC.

SAS Institute was founded in the early 1970s. Two of the founders, Dr. James Goodnight and John Sall, continue to serve as President and Senior Vice President, respectively. The company's flagship product, the SAS Applications System, is an extremely powerful and flexible series of integrated applications that access, manage, analyze, and present data (see Figure 1). The SAS System runs on over 20 different platforms including most UNIX machines, VM and MVS, VMS, PC-DOS and OS/2. The System is currently installed at over 20,000 sites with more than 2 million users. There are 526,000 SAS System products in use on DOS and OS/2 machines alone. Other products the Institute offers are SYSTEM 2000® Data Management Software, the SAS/C® Compiler for S/370s, and JMP® software, a statistical visualization tool for the Macintosh.

Product Summary

Name:	The SAS® Applications System for OS/2
Category:	Applications System
Price:	First year license fee begins at $825 $360 annual renewal. Many discount structures based on quantity. Educational discounts available.
First Ship:	November, 1990
Suggested Memory:	6MB minimum, 8MB recommended
Other Platforms:	IBM Mainframe: MVS, MVS/XA and ESA, VM/CMS, VM/XA and ESA, VSE.
Minicomputers:	VMS (DEC), AOS/VS (DG), PRIMOS (Prime) UNIX - AIX, Ultrix (DEC), HP-UX (HP), SunOS (SUN)
Micros:	DOS, OS/2, Windows 3.0 (December, 1991)
International:	Installed in 90 countries worldwide

MULTIVENDOR ARCHITECTURE

There are many approaches to developing an OS/2 application. One way is to start from scratch, using all of the tools that are native to the environment. This method offers the quickest and most straightforward path to fully utilizing the advanced features of the operating system and Presentation Manager. More planning is required when rewriting an application that is already running on another platform, and might need to be ported to many platforms, to ensure portability and interoperability before writing the first lines of code.

In response to customer demand for transparent cross-platform operation, SAS Institute in 1984 began rewriting the SAS Applications System. Utilizing the C programming language, they developed over 2.5 million lines of *portable* code that comprises nearly 90 percent of the SAS System. Additional code that communicates with individual operating systems or takes advantage of specific features of a platform was then linked in to complete the SAS System for a particular platform. This "MultiVendor Architecture™," as the company calls it, drives the development of the SAS System for all platforms, including OS/2.

The Institute's MultiVendor Architecture (MVA) not only defines the system's portability, but also its modularity. MVA separates the software into three distinct layers: applications, core supervisor, and host supervisor (see Figure 2).

The top layer is the applications layer, which is completely portable and written in C. The applications are the SAS products that can be purchased as add-on modules for functionality such as graphics, applications development, data management and reporting, advanced statistical analysis, and other capabilities. Because the portable code is identical across all platforms, all of the core components of the application are inherently compatible across platforms. It is also the layer where user-written SAS applications reside. The applications code uses no C run-time services, but instead makes calls to the core supervisor. The core layer is portable and developed in C. It provides over 25 subsystems of services for the applications including SAS data set I/O, parsing, tokenizing, and message services. Together the two top layers comprise nearly 85 percent of the SAS System.

At the lowest level, the host supervisor interfaces between the core supervisor and the operating system. The host layer is developed in a combination of C and

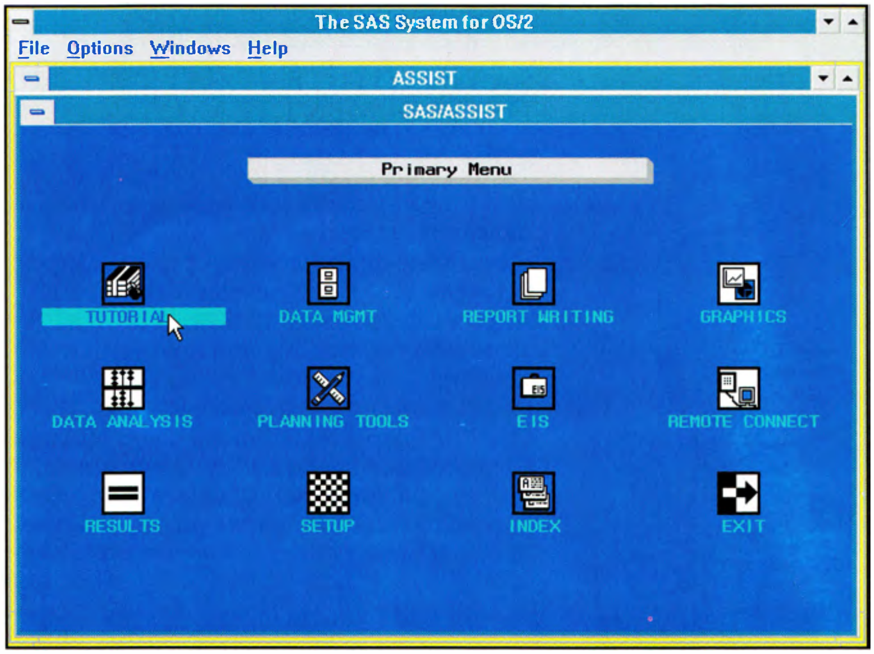


Figure 1. The SAS Applications System provides capabilities for virtually any application that involves accessing, managing, analyzing, or presenting data

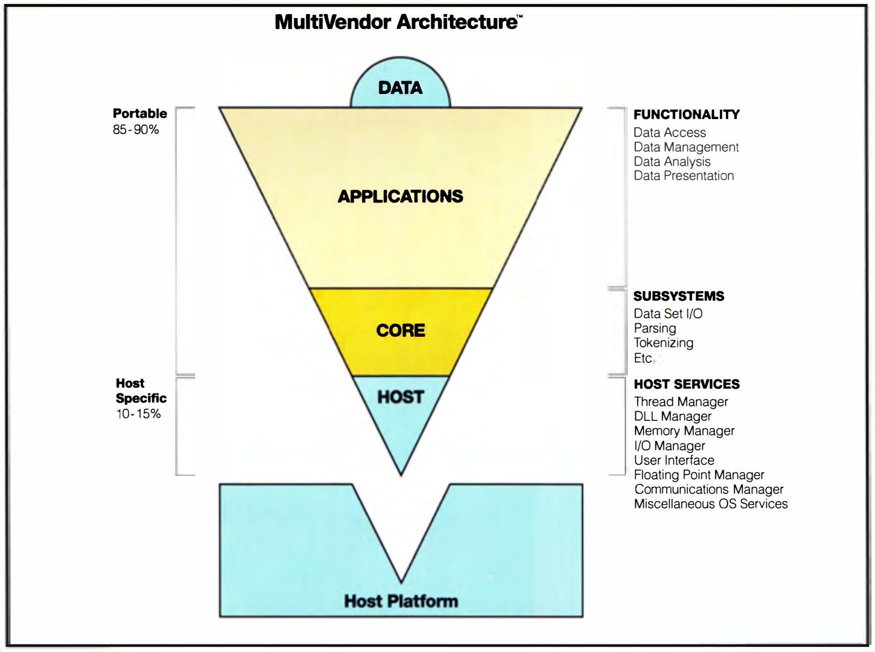


Figure 2. MultiVendor Architecture

Assembler and is not portable. Its job is to take advantage of operating system features and provide performance enhancements to the product. The host supervisor is comprised of 12 subsystems. Some of the major services the host code provides are thread management, DLL management, I/O services, code generation, and the graphical user interface.

SAS INSTITUTE'S OS/2 DEVELOPMENT TEAM

The portable layers of the MVA system are developed by many coordinated software development teams. Most of these teams concentrate on the features of specific product areas. The OS/2 development team is responsible for bringing these source materials together, compiling and linking, and developing the host supervisor code for OS/2. Our discussion took place with the OS/2 development team, managed by Mark Cates. Within his group, the development staff of 14 is further divided into four projects, each responsible for a different aspect of development.

Porting and Building

All host layer code development, including that for OS/2, is actually performed on a network of HP Apollo workstations. All tools (compiler, linker, assembler, and debugger) are provided by the Institute's Compiler Division. These tools for OS/2 run on the Apollo. Source developers on the OS/2 team are responsible for compiling and building the system, then transferring the new DLL files to the PC network on a daily basis. Due to the size of the SAS System, powerful workstations are required for development, because it takes 30 such machines nearly 18 hours to build the entire SAS System and transfer it down to the PC.

Source Managers electronically receive the most current revision of the portable code overnight. During the early stages of development, the portable code is brought down only once every couple of weeks, because it still has many bugs in it. Toward release time, nightly downloads are performed with much cleaner code.

The synchronization of development between the host code and portable code is remarkable. The development of the OS/2 release of the SAS System takes place simultaneously with the development of the SAS System on all 20 platforms. As a result, rigid control mechanisms have been put into place. The groups responsible for the portable code

SAS Institute's OS/2 Development Team. Left to Right, Front to Back: 1st row: Mike Jones, Andy Ju, Carol Williams, Julie Maddox. 2nd row: Mark Cates, Jennifer Clegg, Lynne Smith, Malinda Adams. 3rd row: Don Major, Pat Bostic, Jack Lin. 4th row: Russ Robison, Gary Mehler, John Price, Randy Williams.

continually receive input from the individual host development groups. Each day, the input from all host groups is worked into the portable code. At any time during a host group's development cycle, updated portable code can be downloaded and incorporated into that platform's product.

Although this type of development cycle requires more resources than conventional porting, it is the only way to ensure that an application designed to run identically across multiple platforms truly shares the same source code. This also stabilizes the code quickly, and provides for a very robust system since it is being tested and debugged on many platforms simultaneously.

When porting any source code to a new platform, changes to the source will inevitably have to be made to accommodate specific features of that platform. By having those changes made by the portable source group instead of the host source group, cross-platform compatibility does not have to be sacrificed to take advantage of a specific platform's capabilities. By centralizing modifications to the portable code, future maintenance of it is much easier than if it were heavily modified by individual host groups.

Installation & Product Support

Designed concurrently with the development of the SAS System for OS/2, is an installation program specific to the OS/2 PM environment. The SAS System is a mainframe-size product running on a PC. Because the SAS Applications System is modular, it is important to provide customers with an install program that can selectively load any SAS modules that may be licensed across a combination of local and networked disk drives.

Support for software problems is provided by the Technical Support Division. If any problems are found in the software by customers, patches are provided by the development staff to fix the problems. There is ongoing support for the software ensuring that it is compatible with other OS/2 applications.

Automated Testing

During development an intensive testing project is undertaken. Special care is given to automating as much testing as possible. With hundreds of test programs run nightly, product stability is ensured early in the development cycle. A powerful testing tool, built around the OS/2 API DosPtrace, was created on OS/2 by the testing developers. This tool allows unattended testing to be performed at night. In the morning, OS/2 developers can debug the problems found during the overnight process. In the final phase of the project, the product is released to the Quality Assurance division for final testing and sign-off.

Host Development

The host developers have a dual role. First they must understand the features and functionality of the applications and core levels. Then they must design the interface at the host supervisor level to provide the necessary functionality. Additionally, developers must decide which functions can be provided natively by the operating system and which functions must be developed in the host code.

It is the job of OS/2 host developers to become intimately involved with the functions and features of the operating system. Throughout the cycle, OS/2 developers research operating system features, APIs, and functionality. Then they develop code that allows the SAS System to take advantage of these features. These developers make the SAS System fit into and exploit the OS/2 environment.

The OS/2 rollout was one of our most successful, with $3.1 million revenue in four months!

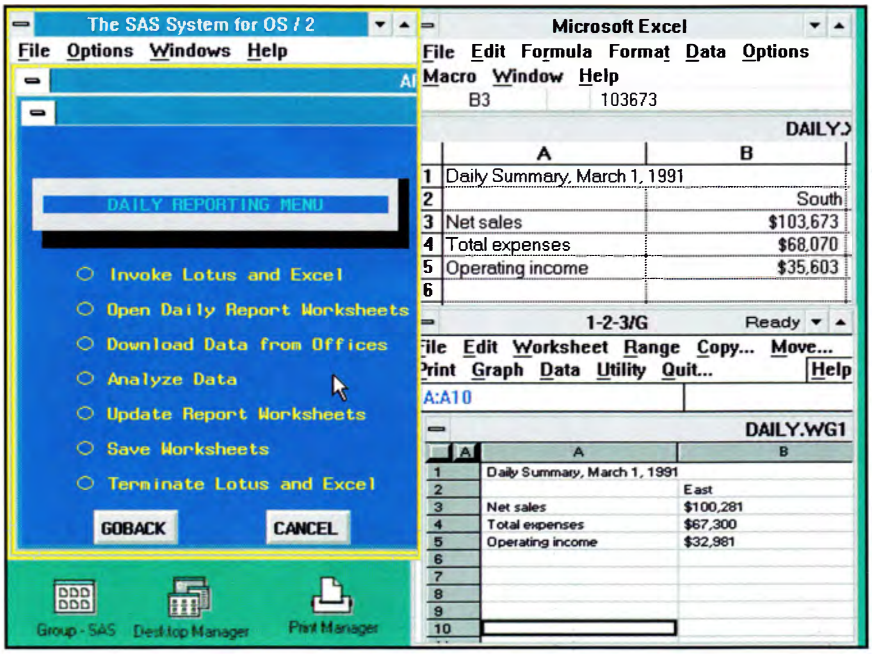
EXPLOITING OS/2 FEATURES

The SAS Applications System for OS/2 exploits many OS/2 and Presentation Manager features, and this helps to make it extremely powerful and flexible. It is the responsibility of the developers of the OS/2 host code to make sure that the SAS System takes advantage of the unique capabilities of OS/2 and its underlying hardware architecture.

Some of the OS/2 and PM features utilized by the Institute's development team are detailed below, along with examples of their functionality:

Named Pipes

Through the use of Named Pipes, the SAS System is capable of establishing many types of communication links. Within one machine, multiple sessions of the SAS System can be running, with the actions occurring in one session controlling the other. Because many of the Institute's customers write their own SAS applications, the use of both Named and Unnamed Pipes allows them to make calls from their own programs to interface with the SAS System.


The SAS System for OS / 2
File Options Windows Help
DAILY REPORTING MENU
○ Invoke Lotus and Excel
○ Open Daily Report Worksheets
○ Download Data from Offices
○ Analyze Data
○ Update Report Worksheets
○ Save Worksheets
○ Terminate Lotus and Excel
GOBACK
CANCEL
Group - SAS
Desktop Manager
Print Manager
Microsoft Excel
File Edit Formula Format Data Options
Macro Window Help
B3
103673
DAILY.X
1-2-3/G
Ready
File Edit Worksheet Range Copy... Move...
Print Graph Data Utility Quit...
Help
A:A10
DAILY.WG1

	A	B
1	Daily Summary, March 1, 1991	
2		South
3	Net sales	$103,673
4	Total expenses	$68,070
5	Operating income	$35,603
6		

A	A	B
1	Daily Summary, March 1, 1991	
2		East
3	Net sales	$100,281
4	Total expenses	$67,300
5	Operating income	$32,981
6		
7		
8		
9		
10		

Figure 3. The SAS Applications System exploits a wide range of OS/2 interprocess communications facilities including Dynamic Data Exchange (DDE), Named and Unnamed Pipes, and the OS/2 Clipboard

The SAS System's implementation of Named Pipes support allows for truly distributed processing in a networked environment. For example, the SAS System for OS/2 could be running on one machine collecting data from a serial data collection device on the factory floor. Via Named Pipes, another machine running the SAS System could receive real-time updates from the first machine for analysis. In this case, the second machine is given access to the acquired data, without actually having any direct link to it. Another advantage to writing to Named Pipes in a client-server architecture is that it allows for a level of independence when running on a local area network. Because Named Pipes support is built into the base OS/2 operating system, communications between machines is independent of the network operating system. As a result, the SAS System supports Named Pipes on any network supporting this feature, including the IBM LAN Server, Microsoft LAN Manager, or Novell® Netware Requester for OS/2.

Cates points out, "The characteristics of Named Pipes make it very exacting, so we added some of our own extensions in order to allow it to be more forgiving to the user. For instance, when you try to connect with Named Pipes and don't get the timing just right, one session or the other might time out and you would have to retry; so we added some configurable options like retry count and wait time to allow the client and server to synchronize more easily."

Dynamic Data Exchange (DDE)

The SAS System for OS/2 also supports DDE. Whereas Named Pipes support is useful for creating links between SAS applications, the SAS System and custom C applications, DDE is useful for establishing communications with other off-the-shelf PM applications. By combining support for both Named Pipes and DDE within the SAS System, an extremely flexible tool is created. An excellent example is given by Andy Ju, Systems Developer.

"Suppose your boss wants you to develop a system that will retrieve the daily sales reports from various branch offices and update several databases. This job involves downloading data from a mainframe, and then updating a SAS data set, a Lotus worksheet, and an Excel worksheet. Projects such as this require the use of several different applications, and you would like to be able to dynamically share data and results among these applications. With DDE, SAS can handle that." (See Figure 3).

Presentation Manager

"The SAS model is not a message-based system", Carol Williams, OS/2 Project Manager explains. "The product we have on OS/2 today is still really a character based system. Its design is like the mainframe products in that it wants to write data directly on the directly on the screen and wants to read characters. PM was particularly challenging because it is message based. Right away we began using multiple threads for synchronization and improved performance. As a PM application you have to read from the input queue every so often or OS/2 gets confused. So before we could begin any PM programming per se, we had to come up with a model that included one thread to satisfy PM and another to satisfy SAS . Then we had to figure out a way to make those two threads communicate and work together transparently so that SAS matched the PM model."

Malinda Adams, Systems Developer, adds, "Threads are a very important feature of OS/2 and a feature we use extensively. Threads are generally required for PM programming but we also utilize threads for communication services, I/O, and control break handling."

Common User Access (CUA)

The Institute has adhered to the concepts of Common User Access (CUA) since before IBM penned the acronym. Just as a consistent user interface across OS/2 applications is extremely important to the success of those OS/2 applications, so is a consistent and flexible user interface across all supported platforms. Explains Cates, "A SAS user may create an application using our Screen Control Language (SCL) and the application will be completely portable across all SAS System platforms from MVS to UNIX X/Windows to OS/2 PM. The application will be the same, but will have the look and feel that the operating system natively provides." (See Figure 4).

OS/2 High Performance File System (HPFS)

Supporting HPFS in itself does not pose any difficulty in writing an OS/2 application, but the Institute's development team pointed out that attention does need to be paid to the support of long file and directory names. Because HPFS supports spaces in both and most other platforms do not, they had to encapsulate names in double quotes in order for them to be interpreted literally as spaces by the portable code.

DYNAMIC LINK LIBRARIES (DLLs)

Because of the large number of DLLs in the SAS System for OS/2 (500 plus), the developers had to come up with an effective way to manage them. According to Pat Bostic, Systems Developer, "We wanted to allow the end-user to install any SAS System module and not have to reboot OS/2 each time. It also did not make sense to keep them all in one location on one fixed disk, so we had to come up with a way that wouldn't depend on LIBPATH for the location of our executables. We decided to use '.' to mean current directory, allowing us to internally change our directory whenever we needed to issue the DosLoadModule that OS/2 requires."

Bostic also pointed out some tradeoffs necessary in using DLLs. "Under OS/2, the operating system performs cascading loads of other DLLs that the original DLL references. If we were not careful upon loading the first SAS DLL, all other DLLs were immediately loaded. We had to carefully define the calling relationships between the DLLs to ensure the cascading loads were not detrimental to SAS performance."

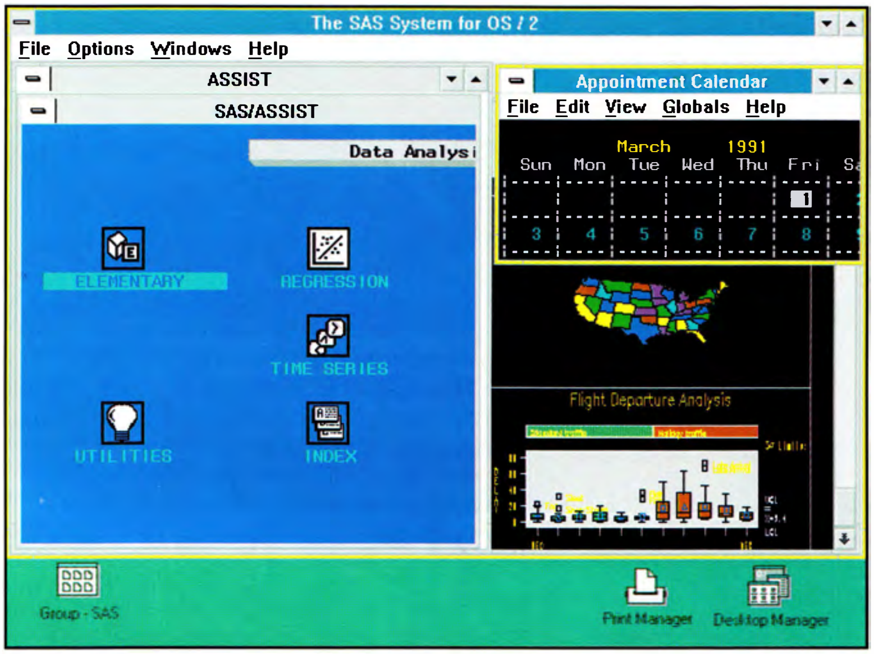


Figure 4. The SAS System's point- and -shoot interface supports CUA standards

One interesting DLL the developers wrote allows users to submit SAS System programs to be processed at preset times. "Through the DosSleep API call, we wrote a standalone DLL that, through the data management part of the SAS System, can submit a job that calls this code and sleeps for a specified period of time, or until a specific date and time. At the specified time, a wake-up function continues the processing of the job where it left off," explained Mike Jones, Systems Developer.

Memory Management

In order to further enhance the performance of the SAS Applications System under OS/2, the development team has written some routines that monitor and react to changes in system memory allocation by the base OS/2 operating system. OS/2 does not have any means of notification when system resources are critically low or when the system is swapping to disk. As a result, as system memory becomes constrained, the performance of the system can slow dramatically. By running a monitoring process on its own thread, the SAS System under OS/2 monitors the memory segment addresses it allocates and determines when thrashing has begun. If it has, the SAS System will automatically unload unnecessary DLLs and those portions of the application that are currently not in use, thus freeing up memory and increasing overall system performance dynamically.

This was not just a quick port from PC DOS to OS/2. It was a major effort to exploit OS/2.

Williams described another enhancement they were able to make in memory management. "Everybody says data structures under OS/2 are limited to 64K due to the Intel segment architecture. Not for us. None of the portable code knows about 64K segment limits, but our compiler knows when accessing memory. An in-line check is made by the compiler to determine when a data structure is crossing a segment boundary. In this way, data structures can be larger than 64K, which is important to large mathematical problems utilizing extremely large matrices."

Cooperative Processing

Through SAS/CONNECT software, the SAS System under OS/2 can communicate to SAS System sessions on the mainframe, mini or Unix platforms. SAS/CONNECT software allows for uploading and downloading of SAS System data sets and catalogs between the machines, taking into account character code translations (EBCDIC to ASCII) as well as numerical precision formats. Additionally, SAS/CONNECT provides for cooperative processing.

"With SAS/CONNECT, you can create a SAS program and test it locally on the PC, and after it passes some test cases, remotely submit the program to the mainframe for a much larger calculation. The results are displayed back in the OS/2 SAS session," says Bostic.

SAS/CONNECT software is very popular on OS/2 and communicates using IBM EHLLAPI and ASYNC interfaces under OS/2 Extended Edition.

Using SAS/CONNECT software, you can also access host data bases by submitting SAS System program statements to a mainframe that accesses a mainframe data base such as DB2. The results of the query will be returned to the OS/2 SAS session for further processing.

DEVELOPMENT PERSPECTIVES

Developer: What were your goals at the start of the project?

Mark Cates

Cates: We had three basic goals. One was making the transformation from a character based user interface to a graphical user interface. Another was to provide a larger, more powerful platform for users. Our customers use the SAS System for some very demanding applications and really push our product hard. This was not just a quick port from PC DOS to OS/2. It was a major effort to exploit OS/2, which brings me to our third goal. The third goal of our development group was to take a lot of pride in taking advantage of everything the operating system had to offer.

Developer: Were there any particularly difficult roadblocks during the porting process?

Cates: One of our biggest portable problems is ints and longs. We have the only 16-bit architectures of all development groups, being responsible for PC DOS, OS/2 and Windows.

Bostic: Our portable code was originally designed for platforms like MVS, where their ints are as large as their longs. When we got down to the PC where ints are only 16-bits and longs are 32, we had problems.

So we introduced extra warnings into the compiler that notify us when we are doing things that could cause those kinds of problems. Being able to modify our compiler to do this gives us an extra level of protection that a standard compiler wouldn't offer.

Williams: That goes for any type of problem we debug more than about three times. We put a warning in the compiler and don't debug it anymore. The portable folks get a warning instead.

Developer: Have the enhancements in OS/2 1.3 made a difference for your application?

Cates: Yes, for us definitely. OS/2 treats network drives as removable media. So whenever we do a DosLoadModule for an image, it loads in that whole image to the local machine. It will start swapping really quickly if you try to do much with it in SAS. OS/2 1.3 fixed that for us. Before, we had to use heuristics in memory management so we could detect if swapping was disabled or SAS was running from a network and warn the user why performance was slow across the network.

OS/2 1.3 was definitely an improvement because of that network issue. Another improvement was that 1.2 had a limitation of the number of windows that could be created. The SAS System ran into this OS/2 limitation, but 1.3 removed it.

Also, we found a lot of improvements in memory management that helped a great deal.

Developer: How do you detect and fix bugs in the software once the product is in the field?

Jones: Our $GPF option helps quite a bit. When a user comes across a bug in our system that causes a general protection fault window to pop up, we can have them bring up the software again using the $GPF option. Special code will be initiated using the DosPtrace API that will write out the name of the application or product, and as much information about the problem as it can get. It will also allow the SAS System to continue running without crashing. I can take that logged information and figure out the code segment and instruction pointer that the GPF occurred at and figure out what happened. It's not always that simple, but it helps a lot.

Developer: Generally, how would you rate OS/2 as an operating system?

Cates: OS/2 is a powerful operating system. It is certainly full featured. It takes a while for developers to become familiar with all the services available, but generally we have found that we can implement most of what we want to accomplish within the API. The level of services provided on OS/2 has made our job easier in some cases, as compared to PC DOS (see Figure 5). One area I would like to see OS/2 improve on is in its critical error recovery and exceptions handling. I think OS/2 needs to provide API hooks and exceptions to inform an application when memory management is thrashing, or when a DLL load could not be accomplished. Give us an exception; sophisticated applications like SAS can benefit from the handlers to produce more robust applications.

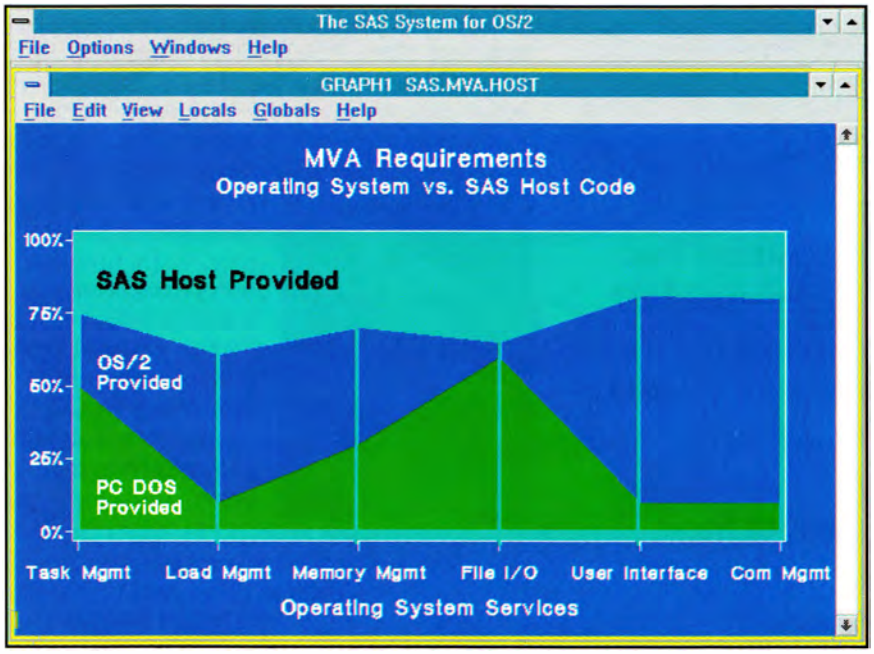


Figure 5. MultiVendor Architecture defines a minimum level of operating system services required by the SAS Applications System. The host development group writes code to supplement the services provided by the particular operating system

MARKETING PERSPECTIVES

We had the opportunity to talk with some of SAS Institute's executive and marketing management to get some insight into the positioning of their OS/2 products. Dr. James Goodnight, SAS Institute's President, was joined by Barrett Joyner, Director of U.S. Marketing, and Randy Betancourt, Manager of Workstations Marketing.

Developer: You are a pioneer in the OS/2 market. Has the investment been worth it?

Goodnight: Our current Version 6 product cost over 100 million dollars to develop, but it is difficult to break out the OS/2 portion of those costs.

> *Our customers were clearly indicating that they wanted us to move to OS/2 from DOS, and that they intended that most of their organizations would be moving to OS/2.*

Jim Goodnight

Joyner: The OS/2 rollout was one of our most successful in history. Right now our users have licensed more than 8000 OS/2 SAS products. In the initial four months, our release for OS/2 has generated about $3.1 million in revenue.

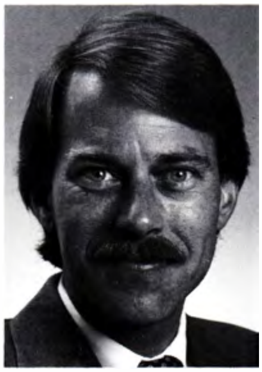
Developer: Who do you see as your primary customers?

Goodnight: Our biggest OS/2 customers are our MVS and CMS shops. That's where we're seeing the majority of our early OS/2 machines going. These are the Big Blue shops that have adopted the IBM concept of using PS/2s and OS/2 as the window to the organization's data and computing resources.

Joyner: A large percentage of our current OS/2 sites are also running the SAS Applications System on the mainframe.

Betancourt: And among our large commercial customers, about 50 percent have also licensed SAS/CONNECT software, our distributed processing product. That suggests a high level of connection between their OS/2 workstations and the host.

Part of our marketing program is to make all of those customers aware of the connectivity strategy that we have for OS/2; to allow it to share in the distribution of SAS System processing from what they already have on the mainframe and give them the ability to bring it down to the desktop. Because our target is our large base of existing IBM 370 and 390 customers, we'll continue to emphasize our connectivity capabilities with OS/2 to the host. This is one of our main messages to those customers.



Barrett Joyner

Developer: Do you expect your OS/2 version to penetrate new markets for the SAS System?

Joyner: A lot depends on the penetration of OS/2 itself. Right now, OS/2 penetration has been primarily in large IBM shops. If OS/2 moves forward into new arenas, we're certainly well positioned to go into them. A number of our customers are actually moving to OS/2 for the sole purpose of using the SAS System on a powerful platform.

Developer:- Was the decision to bring an OS/2 product to market customer or marketing driven?

Goodnight: Our customers were clearly indicating that they wanted us to move to OS/2 from DOS, and that they intended that most of their organizations would be moving to OS/2.

Developer: How much of the product was shaped by technology; by marketing?

Joyner: We're a customer driven company. Because we have such a large and diverse installed customer base, we can go to them and ask what they need or want.

Developer: Which promotions have been the most effective for you for your OS/2 product?

Betancourt: I think the most effective was our distribution of a large number of beta test copies to roughly 300 sites here in the U.S. and about 300-400 in Europe and the

Randy Betancourt

Asia Pacific Rim areas. We were actually one of the first major pieces of software to take advantage of PM that our clients had seen. Another effective marketing tool was our placement of evaluation copies at a number of publications for review. When the reviews came back they were very positive towards the way the SAS System had been implemented in the OS/2 environment.

Developer: Will you be supporting other PC platforms in the future?

Goodnight: We are currently doing very well on our Windows work. It is our plan, over the next year, to release OS/2 and Windows versions simultaneously. They use the same development team, and with much common code between them, we feel that we're always going to keep our releases in sync. We are also committed to OS/2 Release 2.0.

Developer: How do you view the international market?

Goodnight: Especially in Europe, we've received very strong indications that OS/2 is going to be a big item. This year revenues from OS/2 will be split about 50/50 between North America and Europe/Pacific Rim.

Developer: Has the SAS System always been portable?

Goodnight: Version 5, in 1979, was the first to begin being portable because it was designed to run on both IBM and VAX machines. It happened to be written in PL/1 at that time. When the PC platform became our next target, there was no PL/1 compiler. We finally decided that the only really good languages for us were C and Pascal. We knew if we wrote on C, it would be available on almost every platform we needed. The only place it wasn't available was on the mainframe, so when we bought Lattice, we put one there ourselves. We're the number one C compiler for the mainframe, now.

Developer: Who would a developer contact at SAS Institute if they wanted to develop a cooperative application with the SAS System?

Betancourt: I know Mark Cates, Manager of OS/2 and Windows development, and his staff are always working with other vendors testing DDE compatibility and ensuring that the SAS System is compatible with the many ways the product is used. Because of our large presence in the marketplace a number of organizations contact us regarding cooperative applications. If a developer is interested, the Institute can be contacted through Nancy Norman, our Third Party Liaison.

SAS INSTITUTE'S CHECKLIST FOR OS/2 DEVELOPERS

Although not all software companies have the resources of SAS Institute, a lot can be learned from them in terms of strategic planning, resource allocation, and program design. Detailed below is some of the Institute development team's best advice for other OS/2 programmers.

1. Utilize the interprocess communications functions of OS/2, including Named Pipes, DDE, and the Clipboard. The interfaces are built into the OS/2 base system and can allow you to build robust client-server applications that can interact with other PM applications and are completely LAN OS independent.

2. If you ever plan to make your product run on multiple platforms, take the time to separate the OS/2 and PM specific code from the portable code, before you start writing for OS/2 and PM. Even if you don't make the division of code complete, doing so will help speed the development of future versions of your software.

writing for OS/2 and PM. Even if you don't make the division of code complete, doing so will help speed the development of future versions of your software.

3. Keep it modular. Even if you plan on shipping your product in one box, separating the modules will make maintenance of them much easier. Making the system more modular also helps you ship the product sooner. A comprehensive and debugged base system can be put on the market while modules with enhanced capabilities are still in development. Not all applications can be practically designed this way, but it is a good idea when the application can be divided into logical modules. It is also good business. Customers can configure the system to meet today's needs — and can grow with you in the future in a compatible way as their own needs grow and change.

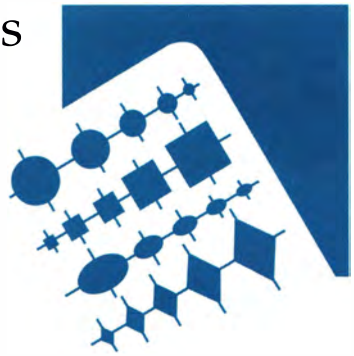
4. Test your product thoroughly before distributing to customers. Although the real world can dictate some strict deadlines for product delivery, too much can be lost in shipping a product with bugs. Shipping a product too early will damage the reputation of the product and will cost your support staff a lot of time (hence money) supporting and patching buggy code.

SAS INSTITUTE INC.

Address:	SAS Campus Drive Cary, NC 27513 (919) 677-8000
Founded:	Incorporated in 1976.
Employees:	2250 Worldwide.
Sales Channels:	Direct in U.S. and 21 other countries. Through distributors in 17 other countries.
First PC Product Shipment:	The SAS Applications System under PC DOS September, 1985.
Other Products:	SYSTEM 2000 Data Management software (IBM mainframes) SAS/C compiler (IBM mainframes) JMP software (Macintosh®).
Developer Contact:	Mark Cates Manager of OS/2 and Windows Host Development

Peter D. Goldstein.
Mr. Goldstein is a consultant to Spectrum in the computer networking industry, and manages the Network Systems Group at EAC Connecting Point in Ridgefield, CT.

Software Tools



New Kids on the Block

by Brian Proffit

This issue's Tools Update column looks at some recently released tools for OS/2.

Life for the OS/2 developer continues to get easier, while simultaneously becoming more exciting. The range of options open to a developer is wider than ever. Not only is it easier to find a tool that is a precise fit to the task at hand, the power and possibilities of today's tools are stimulating the developer's creativity and encouraging more straightforward, intuitive user interfaces.

The success of Windows 3.0 has not slowed development for OS/2. On the contrary, the market's acceptance of GUI (Graphical User Interface) as a concept has stimulated vendors to exploit the enhanced power of OS/2 to bring the development world to new levels.

The three main development tool areas are 1) productivity tools; 2) compilers/ languages;and 3) development environments. Let's look at some new releases in each of these areas and get a feel for the developer's options.

WIDEANGLE™

OS/2 makes it easy for developers to do many things at once. It is not uncommon to have open windows for the File Manager, an editor, a compiler, a debugger, a host session, a to-do list, etc. all simultaneously. The problem becomes how to manage all these windows within the limited real estate of the display. Windows can become totally hidden behind others, requiring the Task Manager to bring them back to the foreground. Frequently minimizing windows to icons and restoring them can be frustrating. Higher-resolution displays help, but there is a limit to the minimum size font that can be read no matter how clearly that font may be displayed.

WideAngle resolves this problem by giving the user up to nine *virtual* displays. WideAngle manages these display areas in which users can position windows. (See Figure 1). The physical display can be switched between the nine virtual images with a click of the mouse.

Brian Proffit

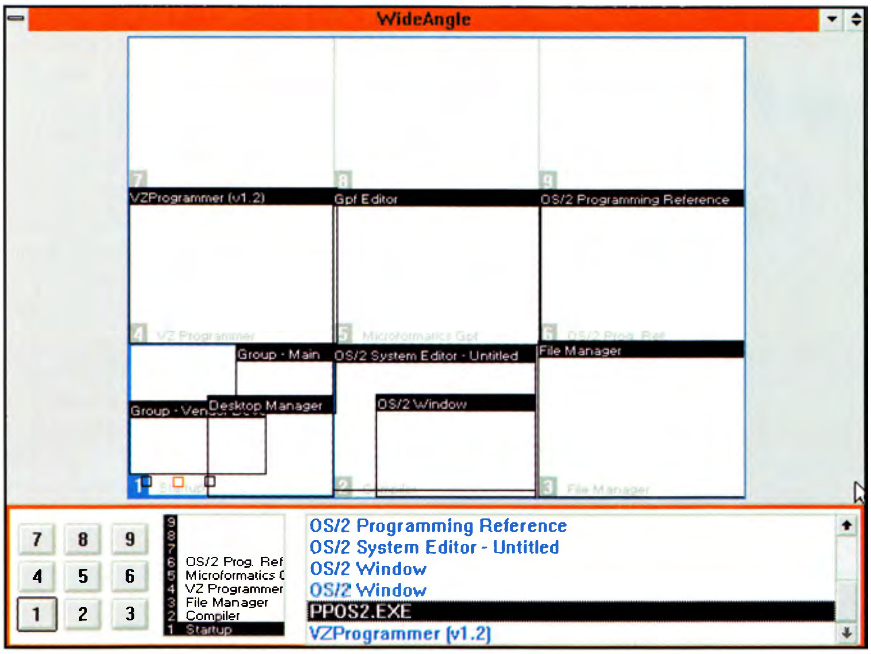


Figure 1. Wide Angle Display Areas

Object-oriented architecture is moving from "an interesting idea" into the mainstream of development.

The maximized WideAngle session shows a representation of all nine virtual displays, and allows the user to move windows anywhere within the area. Windows can even overlap the virtual display boundaries so parts of a window are visible in more than one display group. There is a list box in the WideAngle session with a list of all active sessions, making it simple to move to any of them with a quick click of the mouse.

Many people wear several different hats during a typical day. WideAngle allows you to set up your system for each of your multiple environments. For example, screen 1 may be reserved for the Desktop Manager and the File Manager, allowing the File Manager to take up most of the screen. Screen 2 may be the *development* group, containing an editor, debugger, and command window.

Screen 3 may be the *management* group, containing a to-do list, calendar manager, notepad, and project management software. Screen 4 may contain a spreadsheet, screen 5, a host session, etc. Each screen appears in normal size and is easily accessible without the annoyance of continually minimizing, resizing, and moving windows.

Care to simplify life? Put the commonly-used set of applications in a group under the Desktop Manager named WIDEANGLE. Then add WIDEANGL.EXE to the STARTUP.CMD file. Next morning, turn on the power switch and have all programs loaded with their windows properly sized and positioned without pressing a key.

To summarize, WideAngle should be helpful to anyone with a display less than 36 inches and 8192 x 6044 resolution.

WideAngle is available for $129 from Inner Media, Inc., 60 Plain Road, Hollis, New Hampshire, 03049, or call toll-free, 800-962-2949, (in New Hampshire, 603-465-2990),fax, 603-465-7195.

ZORTECH C++

The world of application development is evolving in several directions, all designed to increase programmer productivity. Object-oriented architecture is moving from its status as *an interesting idea* into the mainstream of development. There are differing opinions on where the object model originated, but C++ emerged as the first widely recognized implementation.

Zortech has been in the C++ business for over five years, and with its version 2.0, the company introduced the first C++ allowing development of full Presentation Manager programs. Now, with its Version 2.1 Developer's Edition, they present a complete development environment.

Working in the Zortech C++ Workbench, the developer has an ANSI C compiler, a C++ compiler, a full-function symbolic debugger, and a large set of tools including such things as class libraries for linked lists and binary tree searches. Using multiple integrated windows, the developer can compile, debug, edit, browse, and use other included tools such as TOUCH and ZGREP.

This is an excellent multi-platform tool, allowing development for DOS, Windows, OS/2, or even SCO® UNIX (with some conditional compilation directives).

Further demonstrating flexibility, the debugging information generated is compatible with Codeview and Multiscope also. The included debugger is quite robust itself, though — with its ability to run in a window and debug multiple threads.

The installation procedure shows a new concept in flexibility. After starting the installation program, the user can insert the diskettes in *any* order! The installation program determines which disk has been loaded and installs the correct information from that disk.

Perhaps the best demonstration of flexibility is the set of class libraries. This is the only package we are aware of that includes the source code for *all* of the class libraries.

By combining an ANSI C compiler with a C++ compiler in the same package, Zortech has given developers the ability to move into the object-oriented world at their most comfortable pace. Use C now, and as you become more comfortable with C++, migrate your applications.

Note that this is a C++ compiler. Many C++ implementations on the market are preprocessors that take the object-oriented constructs and translate them into a form that a native C compiler can understand. With a preprocessor, source level debugging is nearly impossible.

Dynamic link (DLL) modules can be created in C++, and external DLLs can be called from within a C++ program, further contributing to the *reuse of code* concept of object-oriented development. Access to the OS/2-PM API is through the standard headers and libraries in *OS/2 Programming Tools and Information.*

All in all, experienced object-oriented programmers, developers with a C background who are considering movement into the object-oriented environment, or even those just looking for an integrated C development environment will find this to be an excellent choice.

Zortech C++ OS/2 Developer's Edition is available for $600 from Zortech Inc., 4-C Gill Street, Woburn, MA, 01801, 617-937-0696, fax 617-643-7969.

VZ PROGRAMMER™

VZ Programmer is an excellent example of the state-of-the-art in object-oriented development environments. It is a smooth blend of the familiarity of C, the object orientation of C++, a large pre-defined set of classes, and the PM graphical interface.

The most obvious aspect of VZ Programmer is its graphic nature. To create an instance of a push button object, one simply draws it. The built-in class libraries include standard methods for use with these objects, so the user need do no programming to create, size, and draw the object.

VZ Programmer defines most included methods as virtual so they can be overridden, if desired. The iconic toolbox in the development window makes it easy to lay out the user interface with standard PM controls such as scroll bars, push buttons, and list boxes.

A graphic editor is also provided to allow the developer to design boxes, circles, ellipses, etc. in a variety of colors and fill patterns. Although there is no bitmap editor, there is a provision for importing bitmaps developed with other tools, such as the Icon Editor in *OS/2 Programming Tools and Information.*

Accessing all this function is simplified by VZ Programmer's adherence to CUA standards.

As you would expect, each object has attributes (data and/or methods) associated with it. VZ Programmer eases data formatting restrictions by supporting only two forms: string and typeless numeric.

The collection of methods provided is extensive, but most programs will require the developer to modify, extend, or add additional methods. Script attributes fulfill this need.

The Scriptor environment is based on a PM multi-line edit field (as is the OS/2 System Editor), but text files from the developer's favorite editor can be imported also. The script language is VZ C, described as "a full-featured programming language based on the C and C++ programming languages." The set of C library functions is extensive, and there is support for object-oriented extensions, such as a class reference operator "->" and an attribute reference operator "=>".

All child objects of a given parent are maintained in a linked list. This gives a method (script) the ability to traverse the list of child objects using **Parent()**, **First()**, and **Next()** methods. A message can be sent to all objects hierarchically below a given object using the **Broadcast()** method.

The interpretive nature of VZ Programmer means that dynamic link modules can't be developed with it. It is possible, however, to call external DLLs from within a program. This allows the developer to take advantage of VZ Programmer for most parts of a program, while developing pieces that need to be shared across applications or are performance-sensitive in a compiled language.

The success of Windows 3.0 has not slowed development for OS/2.

Development in the Scriptor environment, though, is pleasant enough to tempt one to keep this to a minimum. The interpretive nature, with integrated browsing and symbolic debugging support, is naturally easier than the cycle of edit, compile, study diagnostics, compare compiler line numbers to source, etc.

All this interactivity doesn't come without a price, of course. While the scripts are "compiled" to an internal pseudo-code, all applications must run within the VZ Programmer framework. That brings a certain amount of baggage to an application in disk, memory, and speed. As such, it is likely that VZ Programmer will find its market concentrated more heavily in companies doing internal development than those writing commercial applications.

There is an excellent demonstration of VZ Programmer available on the ESDTOOLS disk, including a detailed tutorial. VZ Programmer is available at a list price of $3,000 from VZ Corp., 57 West South Temple, Suite 210, Salt Lake City, Utah, 84101, 801-595-1352, FAX: 801-328-4404. An unlimited runtime distribution license is $995.

Many people wear several different hats during a typical day. WideAngle allows you to set up your system for each.

SUMMARY

There are excellent new tools being released regularly for OS/2, and they continue to evolve in both form and function. Developers are beginning to think in terms of OS/2 rather than multi-tasking DOS. There are some exceptions, and let me send a brief message to OS/2 developers everywhere: *you can no longer make assumptions about the user's system or environment.*

For example, VZ Programmer will install on any drive, but forces its DLL to be installed onto the C: drive.

Many different packages use an INCLUDE or LIB environment variable. That is understandable, but they insert their path at the front of the user's INCLUDE statement in the CONFIG.SYS file. The developer using many tools is almost sure to come across conflicts between packages. One excellent approach was taken by Microformatics in their Gpf product. It uses a .CMD file that uses OS/2 SETLOCAL and ENDLOCAL statements to set INCLUDE and LIB paths for that session only.

Another frequent offense is insertion of the program's path at the front of the user's LIBPATH. First, this forces additional disk accesses whenever a DLL is loaded from other programs. Second, it forces a reboot of the system after installation. No change should be made to the LIBPATH without the user's consent. Many users now are putting the current directory (".") at the beginning of the LIBPATH so DLLs will be loaded correctly from any application without modification to the LIBPATH.

It is a rare person indeed who buys OS/2 to only run one tool or application. Please maximize flexibility in your installation process and environment requirements.

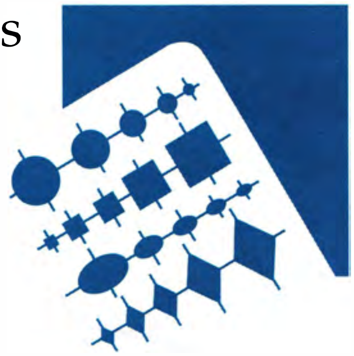
Enough soapboxing. It should be evident from the products reviewed here that the OS/2 developer has a much wider range of possibilities than ever before. From editors to compilers to enablers to development environments.

One last request. Does anyone out there actually know what an *enabler* is? If you have a good definition that distinguishes it from a language, compiler, or development environment, please send it to me on IBMLink at DEV0081 or CompuServe at 75300,1466. Contributor of the best entry will receive an OS/2 tote bag!

Happy Developing.

Brian Proffit, *IBM Entry Systems Division, 1000 NW 51st Street, Boca Raton, FL, 33429. Mr. Proffit is an Advisory Programmer in OS/2 Developer Support Programs. He is currently responsible for OS/2 developer tools. He joined the development laboratory in Boca Raton in 1983 after working as a Systems Engineer in Dallas.*

Software Tools



Checking Memory References

by Robert A. Gibson

Historically, C compilers have been concerned with generating optimized code for the performance-hungry programmer. During development, some of these optimizations can be counter-productive. This article describes a programming technique that uses OS/2 to automatically check for some of the common addressing errors associated with arrays and pointers.

Robert Gibson

Arrays are one of the most commonly used data structures. As such, we frequently have references throughout our programs to different elements of these arrays.

```
#include <stdio.h>
#include <string.h>

#define BUFFER_SIZE 10

void main( void )
{
   char buffer [ BUFFER_SIZE ];
   int i = 65;
   int j = 66;
   int k = 67;

   printf( "Before: %3d %3d %3d\n", i, j, k );
   strcpy( buffer, "----+----1" );       /* 10 bytes plus ASCII zero */
   printf( " After: %3d %3d %3d\n", i, j, k );
   printf( "Buffer: \"%s\"\n", buffer );

}
```

Figure 1. A Program That Copies Too Much Data Into an Array

Each of these references has the potential of being wrong, either syntactically, or semantically. As compiler technology improves, we see compilers becoming more adept at identifying syntax errors. This leaves us with semantic errors that can, for example, cause the modification of totally unrelated variables when our array index is wrong. Programmers familiar with Pascal frequently depend on the compiler to generate array bounds checking. Array indexes that are too small or too large will terminate execution of the code, returning an error message. This checking is rarely implemented in C compilers. The trivial program shown in Figure 1 demonstrates this property.

The output of this program is dependent on the compiler, and possibly on the memory model. Using IBM C/2 Version 1.1, we see the output shown in Figure 2.

Apparently, this particular compiler assigned the first integer, variable i, to a register, and the second integer, variable j, to the memory immediately following the array variable buffer. We can determine this because it is the second value, j, that was modified when the *strcpy* function was told to copy too much data to the target array. The null-termination

```
Before:  65  66  67
 After:  65   0  67
Buffer: "----+----1"
```

Figure 2. Output from Figure 1 Example

character (i.e., the '\0') was copied beyond the end of buffer, and was placed in the low order byte allocated to j.

```
#include <stdio.h>
#include <string.h>

#define BUFFER_SIZE 10

void main( void )
{
   char buffer [ BUFFER_SIZE ];
   int i = 65;
   int j = 66;
   int k = 67;

   printf( "Before: %3d %3d %3d\n", i, j, k );
   strncpy( buffer, "----+----1", BUFFER_SIZE );
   printf( " After: %3d %3d %3d\n", i, j, k );
   printf( "Buffer: \"%s\"\n", buffer );
}
```

Figure 3. Program Limiting the Amount of Data Copied

Of course, we didn't have to use *strcpy*. We could have used the closely-related *strncpy*, which stops copying after the specified length. Figure 3 shows an example of this technique.

```
Before:  65  66  67
 After:  65  66  67
Buffer:  "----+----1B"
```

Figure 4. Output from Figure 3 Example

When compiled and executed, we get the output shown in Figure 4.

Even though *strncpy* didn't modify the adjacent variable, our subsequent reference to buffer as a null-terminated string in the *printf* is, in fact, in error. That is why we get the extraneous 'B' appended to the end of the string. The binary representation of an ASCII 'B' is 66, which is the value of variable *j*. In fact, the only reason that *printf* stopped showing characters after the 'B' was because the second byte of data associated with the value *j* was zero. If the value of j was such that no ASCII zero existed, *printf* would continue displaying characters until an ASCII zero was encountered.

USING DOSALLOCSEG TO ALLOCATE MEMORY

How can we make use of the memory protection inherent in OS/2 to help protect us from ourselves? Instead of using the C run-time for dynamic memory allocation, we ask the operating system for individual chunks of memory, each of which is automatically protected by the hardware.

The program shown in Figure 5 demonstrates the use of the OS/2 function DosAllocSeg for allocating dynamic memory.

The memory allocation routine, GET_MEMORY, returns a **far pointer** to the allocated memory. In order for this pointer to be compatible with the C library routines used by our program (i.e., *strcpy*, and *printf*), we need to specify a memory model (compact, large, or huge) that uses **far** instead of **near** data pointers. With that in mind, we compile and execute our program and get the output shown in Figure 6. When *strcpy* was copying data to the target memory location, it attempted to copy 11 bytes (the 10 character string, plus an ASCII zero) into a memory segment 10 bytes long. At the time of the error, OS/2 shows a full screen pop-up containing system error code **SYS1943**. This code is generated when a process *references memory outside the limits of a memory segment*. In addition to the pop-up, we see the **SYS1811** message, that informs us that it was this program that caused the pop-up to appear.

The hard part is determining what caused the error to occur. In our demonstration program, we see that the first *printf* was able to successfully execute, but no output was seen from the second. Using a source level debugger, or other debugging techniques, we can isolate the error to the *strcpy* statement, where too much data is being copied.

```
#include <stdio.h>
#include <string.h>

#define INCL_DOS
#include <os2.h>

#define BUFFER_SIZE 10
#define MEMORY_ATTR 1      /* Sharing Attribute for Memory Allocation  */

PCH GET_MEMORY( size_t size ); /* function prototype                   */

void main( void )
{
   PCH buffer = GET_MEMORY( BUFFER_SIZE );      /* Get memory from OS/2 */
   int i = 65;
   int j = 66;
   int k = 67;

   printf( "Before: %3d %3d %3d\n", i, j, k );
   strcpy( buffer, "----+----1" );
   printf( " After: %3d %3d %3d\n", i, j, k );
   printf( "Buffer: \"%s\"\n", buffer );
}
/*****************************************************************/
/* GET_MEMORY: Request memory from OS/2, of the specified size.     */
/*****************************************************************/

PCH GET_MEMORY( size_t size )
{
   USHORT rc;         /* Holds OS/2 Return Code            */
   USHORT selector;   /* Segment selector returned by OS/2 */
   PCH dptr;          /* Far pointer generated as a result */

   if ( ! ( rc = DosAllocSeg( size, (PSEL) &selector, MEMORY_ATTR ) ) ){
   OFFSETOF( dptr ) = 0; /* address = selector:0 */
   SELECTOROF( dptr ) = selector;
   } else {
   printf( "Error allocating memory. OS/2 return code = %d\n", rc );
   exit( rc );
   }
   return( dptr );
}
```

Figure 5. Program Using DosAllocSeg to Allocate Dynamic Memory

```
 Before:  65  66  67
SYS1811:

The process has stopped. The
software diagnostic code (trap
number) is 013.
```

Figure 6. Output from Figure 5 Example

What happens if we use *strncpy* to copy in the appropriate number of characters? All we need to change is the *strcpy*, (found in Figure 5) to:

```
strncpy( buffer, "----+----1",
BUFFER_SIZE );
```

After compilation, we execute the program to see the output shown in Figure 7.

From this, we determine that the third *printf* is the one causing the memory reference error. This is because the *printf* attempts to reference memory beyond the end of the memory segment when it tries, unsuccessfully, to find the terminating null. Thus we protect our programs from trying to store too much data, and from attempting to reference memory outside the limits of our array.

```
Before:  65  66  67
 After:  65  66  67
SYS1811:

The process has stopped.
The software diagnostic code
(trap number) is 013.
```

Figure 7. Output from Figure 5 Example Using strncpy

AFTER THE DEBUGGING

After sufficient testing and repair, our program should reach a high level of stability. At this point, it may no longer be cost effective in terms of performance and system resources for our program to continue with this type of dynamic memory allocation. Two options quickly come to mind:

- Modify the memory routines to no longer request OS/2 protected memory.

- Use conditional compilation to use the appropriate memory mechanism.

Each proposal has its merits. However, if we use the latter, our memory allocation and deallocation routines can be more general in nature and are thus candidates for our next project.

Some of the things that need to be considered when developing conditionally compilable code include:

- *Correct function prototypes:* We need to make sure that the appropriate function prototype is generated.

- *Correct header files:* The generated code may make use of different routines, and thus require different header files to be included.

- *Memory model restrictions:* When we remove, or **#undef** the compile-time constant, can our program now use **near** pointers everywhere? This is a potential performance enhancement.

- *Consistent function interface:* The function interface should be the same, independent of the compilation constant's presence or absence. Thus the routines using our function will not be affected if we recompile our function to use the OS/2 memory allocation mechanism, or the C library mechanism.

One example of a conditionally-dependent memory allocation routine is shown in Figure 8.

Using this technique, programmers can specify at compile time whether the program uses protected OS/2 memory, or unprotected C run-time memory. To do this:

- Have the constant **#defined** in your program. However, with this technique the programmer must to remember to go back and either delete the **#define**, or use the **#undef** compile time directive.

- Specify the appropriate compiler directive at compile time. But with this technique it is essential to remember this, and any other compile directives required by the program.

```
/*********************************************************************/
/* GET_MEMORY: When compiled with PROTECT_MEMORY defined, this       */
/* routine requests memory from OS/2. Otherwise, use malloc().       */
/*********************************************************************/

PCH GET_MEMORY( size_t size )
{
#if defined(PROTECT_MEMORY)
USHORT rc;                /* Holds OS/2 Return Code            */
USHORT selector;          /* Segment selector returned by OS/2 */
PCH dptr;                 /* Far pointer generated as a result */

  if ( ! ( rc = DosAllocSeg( size, (PSEL) &selector, 1 ) ) ) {
    OFFSETOF( dptr ) = 0;        /* address = selector:0 */

    SELECTOROF( dptr ) = selector;
  } else {
  printf( "Error allocating memory. OS/2 return code = %d\n", rc );
  exit( rc );
  }
  return( dptr );
#else
  return( (PCH) malloc( size ) );
#endif
}
```

Figure 8. Conditionally Compiled Memory Allocation Routine

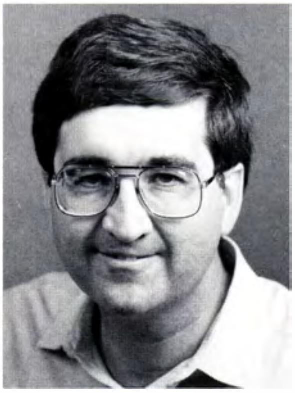
- Have the appropriate constant definition in a *Makefile*. This is probably the most useful technique, because of the prevalent use of the *make* utility to compile all of the appropriate routines and then build the executable module using the correct link invocation. This way, the PROTECT_MEMORY constant can be defined in the *Makefile*, thus clearly documenting to the next programmer how storage protection can easily be enabled or disabled.

Robert A. Gibson, *IBM Corporation, E97/673, P.O. Box 12195, Research Triangle Park, NC 27709. Mr. Gibson is currently a staff programmer in Architecture Compliance, responsible for testing Advanced Program to Program Communications (APPC). He joined IBM in 1977 in Manassas, Virginia, with the Federal Systems Division (FSD). In 1981, he moved to North Carolina, to work on the development of IBM's token ring architecture. He holds a BS in Engineering Science, and an MS in Computer Science from the University of Virginia.*

Extended Edition

Will the Real Client-Server Please Stand Up?

Robert Orfali



Dan Harkey

by Robert Orfali and Dan Harkey

So much has been said about client-server computing that it has come to mean all things to all people. The purpose of this article[1] *is to clarify the client-server terminology and explain what it all means — starting with a definition of the emerging world of client-server computing.*

THE EMERGING CLIENT-SERVER MARKET

Client-server is the culmination of an irresistible trend towards downsizing business applications to PCs, PS/2s®, Macintosh® and UNIX workstations. Downsizing is a process which breaks up large supermini type applications into program modules which run on one or more network servers. The user-interface functions are moved to the client workstations with the intention of replacing the *green screen uglies* with state-of-the-art graphical user interfaces.

Client-server software solutions on low-cost standard hardware are the driving force behind downsizing. Client-server allows us to create a coherent environment out of autonomous desktop workstations. It combines the best of two worlds: the cost effective and almost addictive power of desktop computers with multi-user access to shared resources and data. Client-server solutions almost always result in lower development and maintenance costs. They also provide an open flexible environment where mix-and-match is the rule. Through the SAA™ architectural context, client-server may also provide the seamless integration of PCs and mainframes.

It's no wonder that Forrester Research Inc. projects the client-server market will account for $29 billion worth of sales by 1993. The big losers from these sales would be minicomputer-based solutions.

WHAT IS A CLIENT-SERVER?

Even though *client-server* is a leading industry buzzword, there is no agreed upon definition of what that term means. This provides us with the opportunity to come up with our own definition. As the name implies, clients and servers are separate logical entities that work together over a network to accomplish a task. What makes client-server different from other forms of distributed software? We propose that all client-server systems have the following distinguishing characteristics:

- **Service:** Client-server is primarily a relationship between processes running on separate machines. The server process is a provider of services. The client is a consumer of services. In essence, client-server provides a clean separation of function based on the idea of service.
- **Shared Resources:** A server can service many clients at the same time and regulate their access to shared resources.
- **Asymmetrical Protocols:** There is a many-to-one relationship between clients and the server. Clients always initiate the dialog by requesting a service. Servers passively wait on requests from the clients.

[1] Orfali, Robert and Harkey, Dan. *Client-Server Programming with OS/2 Extended Edition*, 1991 by Van Nostrand Reinhold. (Reprinted by permission of the publisher. All rights reserved.) This book may be ordered from IBM Mechanicsburg as publication G325-0650, or purchased in a local bookstore.

- **Transparency of Location:** The server is a process which can reside on the same machine as the client or on a different machine across a network. Client-server software usually masks the location of the server from the clients by redirecting the service calls when needed.

- **Mix-and-Match:** The ideal client-server software is independent of hardware or operating system software platforms. Users should be able to mix-and-match client and server platforms.

- **Message-Based Exchanges:** Clients and servers are loosely coupled systems which interact through a message-passing mechanism. The message is the delivery mechanism for service requests and replies.

- **Encapsulation of Services:** The server is a *specialist*. A message tells a server what service is requested and the server then determines how to get the job done. Servers can be upgraded without affecting clients, as long as the published message interface is not changed.

- **Scalability:** Client-server systems can be scaled horizontally or vertically. Horizontal scaling means adding or removing client workstations with only a slight performance impact. Vertical scaling means migrating to a larger and faster server machine or to multiservers.

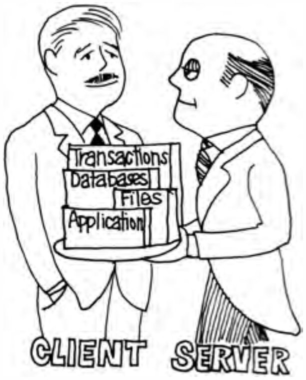
- **Integrity:** The server code and server data is centrally maintained which results in cheaper maintenance and the guarding of shared data integrity. At the same time, the clients remain personal and independent.

The client-server characteristics described above allow intelligence to be easily distributed across a network and provide a framework for the design of loosely coupled network-based applications.

WILL THE REAL CLIENT-SERVER PLEASE STAND UP?

Many systems with very different architectures have been called *client-server*. System vendors often use *client-server* as if the term can only be applied to their specific packages. For example: file server vendors swear they first invented the term, and database server vendors are known in some circles solely as the client-server vendors. So who is right? Which of these packages is the true client-server? The answer is *all of the above*.

The idea of splitting an application along client-server lines has been used over the last ten years to create various forms of local area network software solutions. Typically these solutions sell as shrink-wrapped software packages, and many are sold by more than one vendor. However, each of these solutions is distinguished by the nature of the service it provides to its clients as shown in the examples which follow.



FILE SERVERS

With a file server the client (typically a PC) passes requests for file records over a network to a file server (see Figure 1). This is a very primitive form of data service that necessitates many message exchanges over the network to find the requested data.

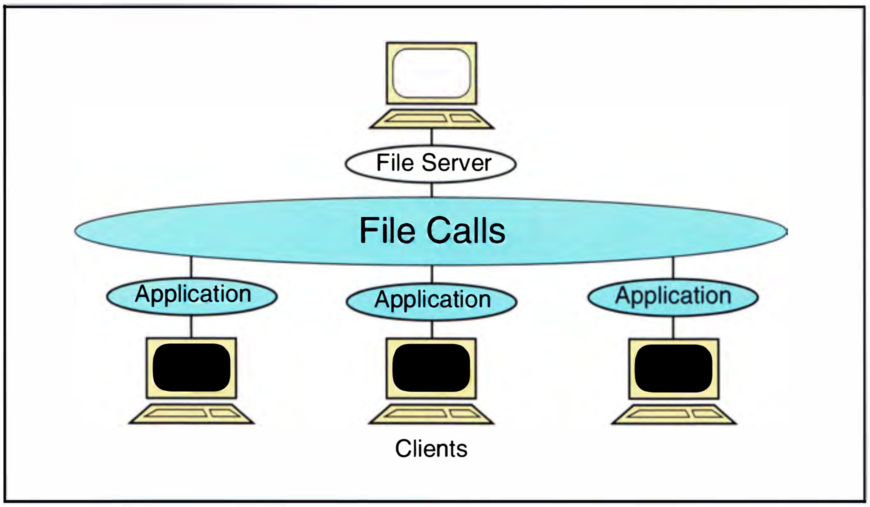


Figure 1. Client-Server with File Servers

DATABASE SERVERS

With a database server, the client passes SQL requests as messages to the database server (see Figure 2). The results of each SQL statement are returned over the network. The code that processes the SQL request and the data reside on the same machine. This allows the server to use its own processing power to find the requested data — as opposed to file servers, which pass all the records back to a client which must find its own data. The result is a much more efficient use of distributed processing power. With this approach, the server code is shrink-wrapped by the vendor, but developers often need to write code for the the client application. (Shrink-wrapped client packages like Query Manager or Paradox are also available.)

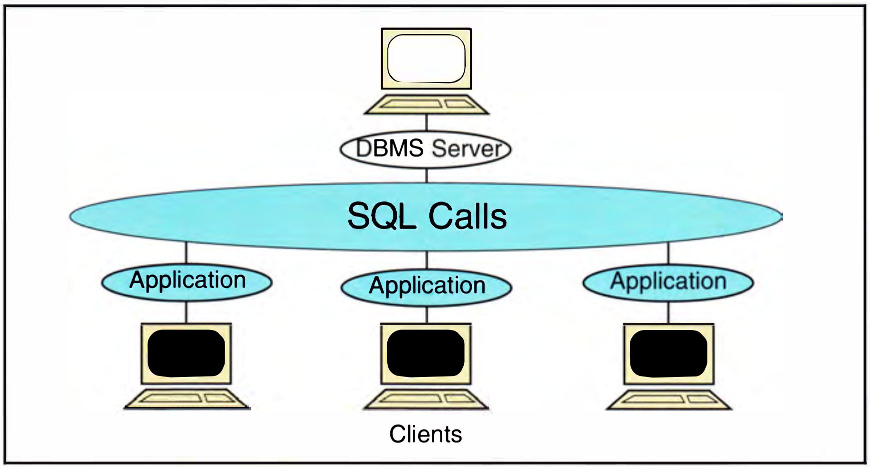


Figure 2. Client-Server with Database Servers

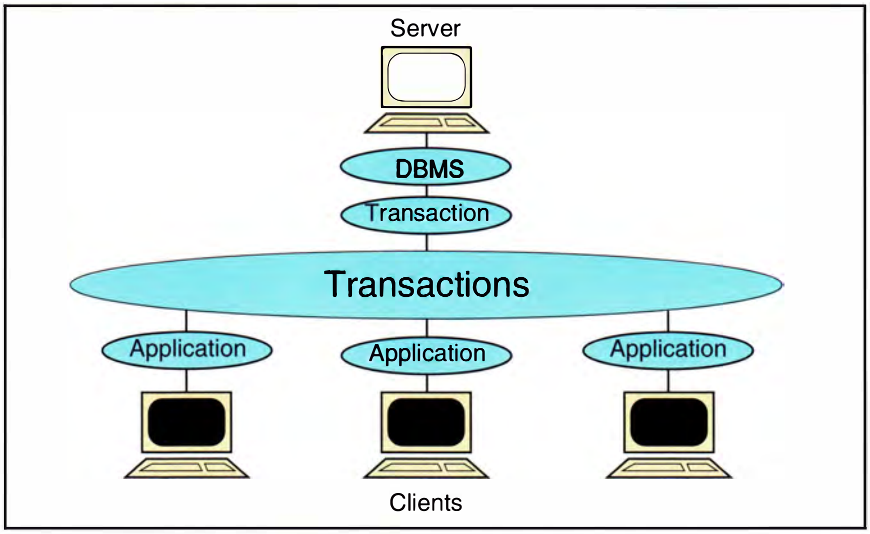


Figure 3. Client-Server with Transaction Servers

TRANSACTION SERVERS

With a transaction server, the client invokes remote procedures that reside on a server which also contains an SQL database engine (see Figure 3). These remote procedures on the server execute a group of SQL statements. The SQL statements either all succeed or fail as a unit. These grouped SQL statements are called transactions. Client-Server applications are created by writing the code for both the client component (which usually includes a graphical user interface) and the server transactions. These applications are called *On-Line Transaction Processing* or OLTP. They tend to be mission-critical applications which require a 1-3 second response time 100 percent of the time. These applications also require tight controls over the security and integrity of the database. The communication overhead in this approach is kept to a minimum. The exchange consists of a single request/reply (as opposed to multiple SQL statements). Programmers must use a peer-to-peer protocol to issue the call to the remote procedure and obtain the results.

APPLICATION SERVERS

With an application server, programmers must supply the code for both the client and the server (see Figure 4). However, unlike transaction servers, application servers are not necessarily database centered. For example, an application server can control a device such as an optical juke-box, or serve as an inference engine with download capabilities from Dow Jones. An application server is a general purpose roll-your-own network application that uses the client-server model.

Application and transaction servers represent the new growth area for PC-based client-server computing. They both require programmers to create custom applications that use the client-server model and build on top of *shrink-wrapped* system components. The idea is to create vertical packages that provide total business solutions using low-cost commodity type hardware and Local Area Networks.

WHAT OS/2 BRINGS TO CLIENT– SERVER

Client-server solutions on a PC platform require the services of a robust operating system which provides multitasking, a multi-user database engine, LAN communications, and a graphical user interface. The server should be able to handle concurrent requests from clients with minimal performance degradation. Client-server solutions on PCs must provide the same level of service as multi-user superminis while maintaining the autonomy of the client PCs. This is where OS/2 Extended Edition® (OS/2 EE) comes into the picture:

- OS/2 EE servers provide minicomputer power at PC prices. OS/2 EE running on a PC with an 80486 processor has the power of a minicomputer in terms of both brute processor speed and system software sophistication (e.g. built-in SQL Database Manager). OS/2's preemptive multitasking and multithreading services, and its memory protection make it possible to create *mission-critical* systems around personal computers.

- OS/2 is an *open* commercial platform. An open platform provides a choice of vendor products and services at competitive prices. This translates into enormous savings in hardware and software system costs when compared to equivalent supermini solutions.

- OS/2 EE servers, unlike minicomputers, were designed from the ground up to seamlessly communicate with IBM PC DOS, MS-DOS® and OS/2 clients. The client and server use the same files, commands, and internal data representations. Operating system calls can be transparently redirected to the server via a network operating system.

- OS/2 EE eliminates the need for minicomputers to act as the middle tier between PC clients and hosts in three-tiered networks. OS/2 EE is fully equipped to seamlessly communicate with SAA mainframes.

In short, OS/2 EE is a commercially available operating system that comes fully loaded with the hooks that make it possible to design client-server systems around personal computers.

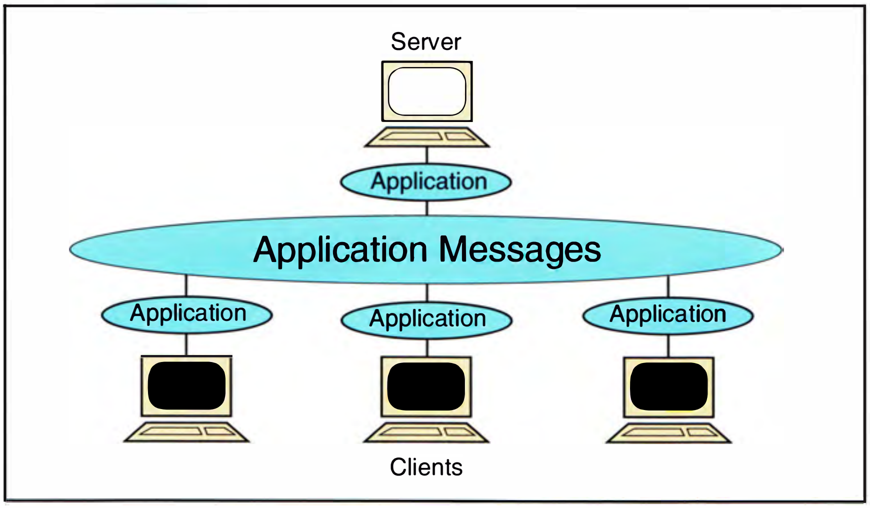


Figure 4. Client-Server with Application Servers

REFERENCES

Orfali, Robert and Harkey, Dan. *Client-Server Programming with OS/2 Extended Edition* (New York: Van Nostrand Reinhold, 1991). This article is an excerpt from Chapter 1. This book may be ordered from IBM Mechanicsburg as publication G325-0650.

Robert Orfali, *IBM Corporation, 5600 Cottle Road, San Jose, CA 95193, is an Advisory Engineer in IBM System Storage Products Division, Technical Applications Development. He joined IBM in 1972 and is the architect of TxE, a client-server "Transaction Enabler" platform. Mr. Orfali is co-author of* ***Client-Server Programming with OS/2 Extended Edition*** *and received an MSEE degree from the University of California at Berkeley.*

Dan Harkey, IBM Corporation, *5600 Cottle Road, San Jose, CA 95193, is an Advisory Engineer in IBM System Storage Products Division, Technical Applications Development. He joined IBM in 1977 and is the lead developer of TxE, a client-server "Transaction Enabler" platform. Mr. Harkey is co-author of* ***Client-Server Programming with OS/2 Extended Edition*** *and received an MSCS degree from Santa Clara University.*

Extended Edition Communications Manager

Classic Client-Server Transactions Using APPC

Lance D. Bader

John Q. Walker II

by Lance D. Bader and John Q. Walker II

APPC (Advanced Program-to-Program Communications) is an application programming interface for communicating programs, allowing them to exchange information with similar programs on a Systems Network Architecture (SNA) network. In OS/2, it plays a central role in the Communications Manager and in IBM SAA Networking Services/2.

APPC can be seen as a toolkit for building applications that are distributed across a computer network. Unlike other communications protocols that only provide a means of sending data across a network, APPC also offers assistance in distributing the control and flow of a transaction among the programs in the network.

Five classic transactions and their motivations are discussed. For each of these transactions, we will show the correct sequence of the APPC verbs for your programs to issue.

APPC TRANSACTIONS

What is a transaction? Definitions vary, but we like to define a transaction this way.

- A distributed application is an application that requires programming from different places.

- A transaction program is one of the programs in a distributed application.

- A transaction is a business deal cooperatively completed by two or more transaction programs.

Note that the term *places* is intentionally vague. Usually we think of applications distributed across a network, but it is just as valid to think of applications distributed across an I/O bus or a process boundary, or even between windowed PM applications.

Likewise, the term *business deal* is intentionally vague. The details of the deal depend on the way the distributed application organizes its work. The deal could be as simple as a message exchange or as complicated as state transitions during a space shuttle launch.

A key aspect of a transaction is this: a transaction has a well-defined beginning and a well-defined end. This allows the systems that support the transaction programs to be clever when it comes to managing system resources (such as program memory, data communication buffers, and transport pipes).

Instead of permanently allocating the system resources to an application, the system keeps the resources in a common pool. When a transaction begins, resources are removed from the pool to service that transaction. When the transaction ends, resources are returned to the pool so that they can be reused for other transactions. The result is an effective utilization of system resources. It was the popularity of transaction programming that fueled the invention of APPC.

Transaction programs inevitably exchange more than data. They must also exchange information about the state of the transaction. Here are some examples:

- This is a new transaction.
- This transaction is complete.
- This program would like permission to send.
- This program grants that program permission to send.
- This program wants confirmation that all the data sent so far has been received and processed successfully.
- This program confirms its success. The transaction programs are now synchronized.
- This program wants to report an error and negotiate the recovery.

The APPC programming interface lets transaction programs easily exchange state information, as well as data. All the code necessary to manage this information (for example, resource assignment, special record formatting, protocol handling, race resolution, and buffer management) is provided by this common interface. This leaves you, as a transaction programmer, free to focus on the specific code necessary to perform your transaction, rather than duplicating the code to exchange state information in every application.

So, how do you write a useful transaction using APPC? There are a lot of transactions in the world already, but a significant majority of the transactions that do real work can be boiled down to one of the five examples shown in this article. You will see these classic transactions frequently when you look at distributed applications. Lots of programmers are writing transactions like these, even programmers who are not using APPC.

You can find more information on APPC concepts and operation in the *Transaction Programmer's Reference Manual* (reference 2) and *APPC: Introduction to LU 6.2* (reference 3). *The APPC Programming Reference* (reference 1) specifically describes how to use APPC with OS/2 EE.

THE BUILDING-BLOCKS OF A TRANSACTION: APPC VERBS

To request APPC to take some action for your program, your program issues a *verb*. A verb is a formatted request that is executed by the APPC software supporting your program. Your program passes a structured list of parameters to APPC, which executes the verb and returns information to your program.

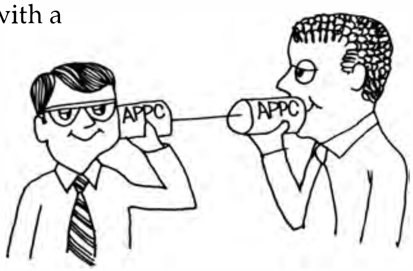
What are the APPC verbs? Think of the tasks that your program needs to perform to carry on a conversation with another program.

Allocating a Conversation

Somewhere in the network is a transaction program that requires assistance to complete a transaction. Somewhere else, there is a transaction program that can provide that assistance. The first program starts a conversation to the second.

Let's call the first program the *client*, and second program the *server*. (The transactions in this article can be used as a template for designing transactions that fit in this client-server model.)

If your program is a client program, it will need to allocate a conversation with a server program. To do this, it issues an ALLOCATE verb.



Sending Data

Your program may need to send data. It uses the SEND_DATA verb to send a record. Do not expect the internal APPC software to transmit the data immediately. APPC pools its internal resources, trying to reduce the overhead of each conversation. It places your data into an internal buffer and returns control to your program. APPC will later transmit the internal buffer when your program sends more data than will fit in the internal buffer, or when your program makes it clear that the data should be transmitted.

For example, your program makes it obvious that it has finished sending by receiving data or deallocating the conversation.

Receiving Data

Your program may need to receive data. Obviously, if one program sends data, the other program must be able to receive it. APPC makes that as painless as possible.

Your program issues a RECEIVE_AND_WAIT verb, asking APPC to wait for all the data to arrive. If the memory segment your program provides is large enough to receive all the data, APPC returns the data and informs your program how much data was returned. If your buffer is not large enough, APPC fills your program's buffer and keeps the rest of the data so that it may be received with the next request.

Receiving data from the other program is similar to reading sequential records from a disk file. Normally, your program does not read a specific number of records. Instead, it reads records until the an end-of-file indication is returned.

With APPC, the same logic is used. Unless your program reports an error, it should continue to receive data records until APPC informs it that there are no more records to receive. To design your programs to be as robust as possible, avoid logic that reads only a specific number of records.

APPC offers assistance in distributing the control and flow of a transaction among the programs in the network.

Granting Permission to Send

Your program may need to grant permission to send. An APPC conversation enforces a *two-way alternate* data transfer (also known as *half-duplex flip-flop* data transfer by communications programmers). This means that at any point in time, only one program has permission to send data. This really reduces the complexity of your programs by eliminating the need to write code that asks things like "Did the other program send this record before or after receiving the last record my program sent?"

When the conversation is allocated, the client program is initially given permission to send. When a program with permission to send is finished sending data and needs to receive the reply, it grants the other program permission to send. In this fashion, permission to send alternates between the programs. The RECEIVE_AND_WAIT verb serves the dual purpose of both receiving data and granting the partner the permission to send.

Requesting a Confirmation

Occasionally your program will reach a point where it cannot continue until it knows that the other program has received and successfully processed all the data that has been sent so far. For example, consider a program that moves data records from one database to another. The program cannot delete the records from the first database before it has been assured that the other program has successfully entered the records into the second database.

When your program requests a confirmation, using, for example, the DEALLOCATE verb with type(SYNC_LEVEL), APPC forces all the previously sent data in its internal buffer to be transmitted to the other program. After the other program has processed all the data records, it will be informed that your program has requested a confirmation. The other program will grant the confirmation if the data records were processed successfully or reject the confirmation otherwise.

Granting or Rejecting a Confirmation

Your program may need to grant or reject a confirmation. Consider a scenario where the other program requested a confirmation from your program. When APPC informs your program of the confirmation request, your program should complete any unfinished processing of the previously-received data and determine how to respond to the request. If the processing is successful, the confirmation should be granted, using the CONFIRMED verb, for example.

By issuing the CONFIRMED verb, the server promises that it has done everything that was requested. The data has been safely captured and locked away; it is not lying in cache. For example, when using the OS/2 High Performance File System (HPFS), make sure that the buffers have been flushed to disk before issuing the CONFIRMED verb.

Similarly, issue CONFIRMED after a successful close of a data file, not before. If the machine crashes after you issue this verb, the data is safe.

If the processing was not successful, the confirmation is rejected by reporting an error. Additionally, your program may detect an error before APPC informs your program of the confirmation request. If your program reports the error, APPC will treat it as a confirmation reject and inform the other program appropriately. (This is just one of the many race conditions that APPC handles in a way that does not have an impact on your program.)

Ending a Conversation

The transaction programs will need to end their conversation. Either program may end the conversation. APPC will inform the other program that the conversation has been *deallocated*, and release any network resources. Issue the DEALLOCATE verb to initiate termination of a conversation. In OS/2, your program can also the type parameter of the SEND_DATA verb to send data and deallocate the conversation with one single verb.

Notes on Reading the Example Figures

These examples cover fundamentals; they do not show some of the advanced APPC functions such as SEND_ERROR or PREPARE_TO_RECEIVE. However, they illustrate how much can be done with APPC without worrying about its advanced functions.

These examples only show the normal processing portion of the transaction, without error handling. They will run as is, but they contain no recovery. If recovery is important (or rather, when recovery becomes important to you), you will need to add that code.

APPC has two flavors of verbs, basic and mapped. Because mapped verbs are somewhat easier to use, they are shown in the figures that follow. They have the same names as basic verbs, distinguished by an "MC_" preceding their name.

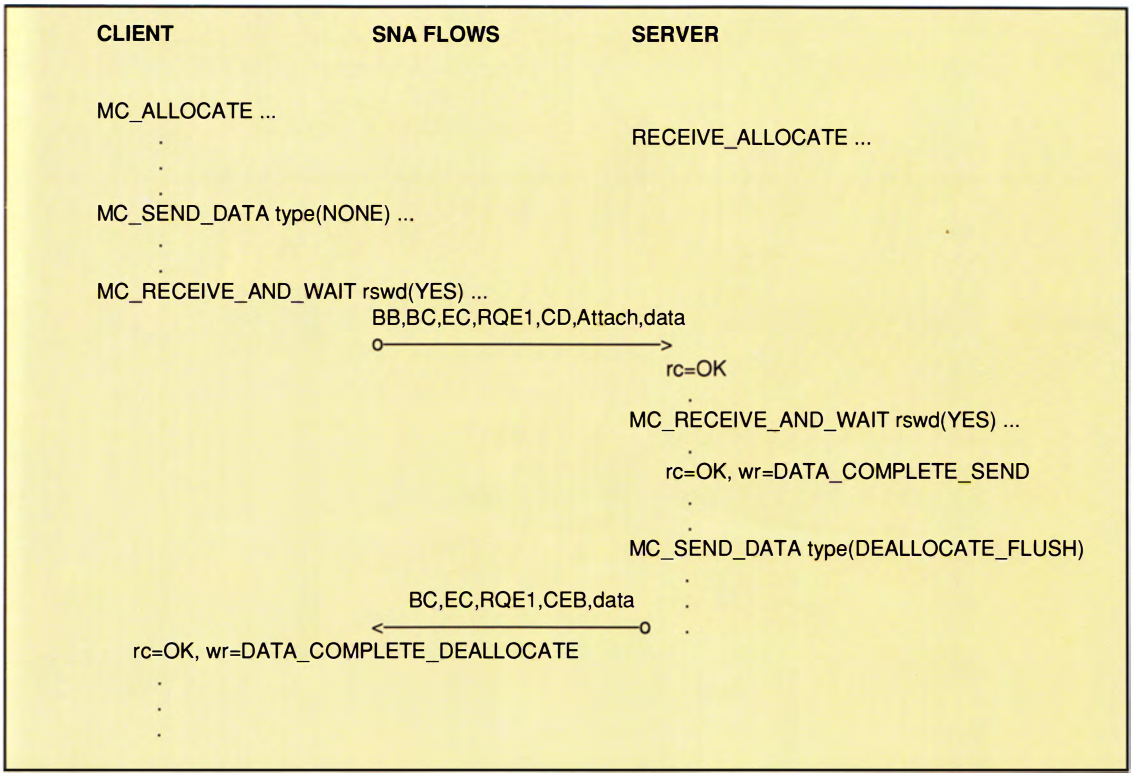
In the example figures, "rc=" indicates the return code value, and "wr=" indicates the returned what_received value.

We have shown the actual SNA line flows in these examples. If you are not interested in these, you can comfortably ignore them. They are included for completeness.

THE INQUIRY TRANSACTION (REQUEST AND REPLY)

The inquiry transaction is typically used to request information from a server program. This transaction is also known as *request and reply*, because the entire conversation is

A significant majority of the transactions that do real work can be boiled down to one of the five examples shown in this article.



```
CLIENT                        SNA FLOWS                    SERVER

MC_ALLOCATE ...
      .                                                    RECEIVE_ALLOCATE ...
      .
      .
MC_SEND_DATA type(NONE) ...
      .
      .
MC_RECEIVE_AND_WAIT rswd(YES) ...
                              BB,BC,EC,RQE1,CD,Attach,data
                              o--------------------------->
                                                             rc=OK
                                                             .
                                                           MC_RECEIVE_AND_WAIT rswd(YES) ...
                                                             .
                                                             rc=OK, wr=DATA_COMPLETE_SEND
                                                             .
                                                             .
                                                           MC_SEND_DATA type(DEALLOCATE_FLUSH)
                                                             .
                                 BC,EC,RQE1,CEB,data         .
                              <--------------------------o   .
   rc=OK, wr=DATA_COMPLETE_DEALLOCATE
      .
      .
      .
```

Figure 1. Inquiry (Request and Reply) The return_status_with_data (rswd) parameter (resulting in combined what_received indicators) is used on the MC_RECEIVE_AND_WAIT verbs, and the type parameter is used on the MC_SEND_DATA verbs, in order to minimize the APPC verb overhead.

made up of one request and one reply. A client sends a request to a server, which computes the answer and replies.

You will find inquiry transactions frequently used by banks and retailers. This type of transaction is good for the following situations:

- *Remote Procedure Calls:* A client program sends a subroutine call request to a server program. The server program performs the subroutine and returns the result.
- *Database Applications:* A client program sends a set of keys to the database server, which then returns the data record selected.
- *Status Queries:* A client program wants to know "Are you there, and what is your status?"

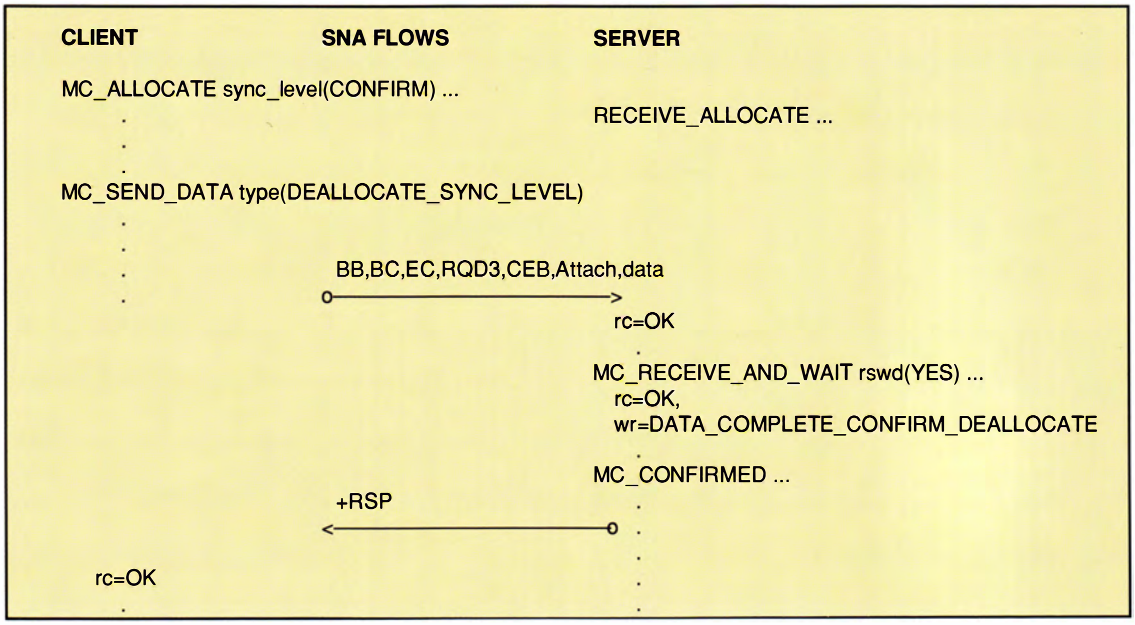
In Figure 1 you will find pseudocode that implements this transaction using mapped conversation verbs. Several parameters for these verbs are not shown (such as the buffer parameters), to emphasize the essence of the transaction.

THE CREDIT CHECK TRANSACTION (CONFIRMED DELIVERY)

The credit check transaction is used when a client program must request permission to perform a specified function, and the probability that the server will grant permission is much larger than the probability that the server will not grant permission. This transaction differs from the inquiry transaction in that no data is returned by the server; the server just returns an acknowledgement that the data sent by the client program was successfully received and processed.

To understand the usefulness of this transaction, consider a distributed application designed to do credit checks. The client program builds a credit check record (which includes an authorization code to be returned to the salesperson if the purchase is authorized), sends the record to the credit check server, and requests a confirmation.

If the credit check is OK (the usual case), no reply is necessary; the server only needs to grant the confirmation. If the credit check is not successful (a rare case), the server rejects the confirmation by reporting an error and sends an error record that explains why the credit check failed. Notice that in the usual case, only a confirmation returns to the client.



```
CLIENT                        SNA FLOWS                  SERVER

MC_ALLOCATE sync_level(CONFIRM) ...
      .                                                  RECEIVE_ALLOCATE ...
      .
      .
MC_SEND_DATA type(DEALLOCATE_SYNC_LEVEL)
      .
      .
      .                       BB,BC,EC,RQD3,CEB,Attach,data
      .                       o------------------------>
                                                           rc=OK
                                                           .
                                                         MC_RECEIVE_AND_WAIT rswd(YES) ...
                                                           rc=OK,
                                                           wr=DATA_COMPLETE_CONFIRM_DEALLOCATE
                                                           .
                                                         MC_CONFIRMED ...
                              +RSP                         .
                              <------------------------o   .
                                                           .
   rc=OK                                                   .
      .                                                    .
```

Figure 2. Credit Check (Confirmed Delivery).

Because a reply does not have to be formatted, sent, or received, the overhead and turnaround time is minimized. On the other hand, the error case creates an additional flow in addition to the reply. This is an acceptable trade-off if the error case is rare enough. The server does not have to send an explicit reply if the transaction can be processed successfully. If this case occurs frequently enough (a rule of thumb is 80 percent of the time or better), the credit check transaction performs better than the inquiry transaction.

This type of transaction is also good for the following situations:

- *Confirmed Delivery:* A client program needs to send information to the server and needs a guarantee that the information has been successfully received and processed.

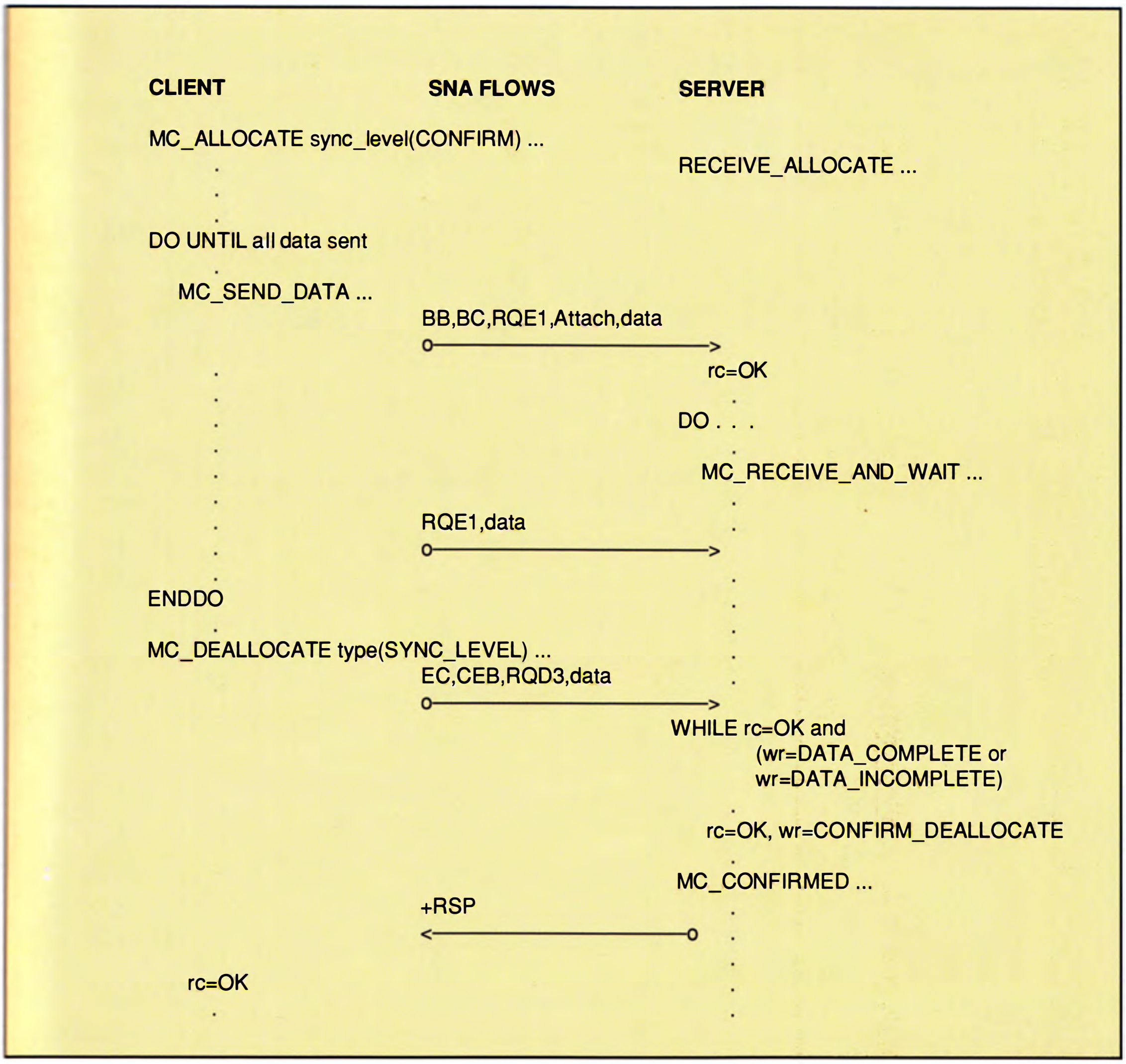
- *Permission:* A client program needs permission from a controlling server before it can continue.

- *Polls:* This is used for applications wanting to know simply "Are you there?" The reply indicates "Yes, I am here."

In Figure 2 you will find pseudocode that implements this transaction using mapped conversation verbs.

THE FILE TRANSFER (BATCH SEND)

This transaction is an extended version of the credit check transaction. This is the first place where we show the buffering capabilities of APPC. Using APPC, your programs do not need to be concerned about record sizes to optimize links and



```
CLIENT                             SNA FLOWS                    SERVER

MC_ALLOCATE sync_level(CONFIRM) ...
        .                                                       RECEIVE_ALLOCATE ...
        .
        .
DO UNTIL all data sent
    .
    MC_SEND_DATA ...
                                   BB,BC,RQE1,Attach,data
                                   o-------------------------->
        .                                                           rc=OK
        .                                                           .
        .                                                       DO . . .
        .                                                           .
        .                                                          MC_RECEIVE_AND_WAIT ...
        .                                                           .
        .                          RQE1,data                        .
        .                          o-------------------------->
        .                                                           .
ENDDO                                                               .
        .                                                           .
MC_DEALLOCATE type(SYNC_LEVEL) ...                                  .
                                   EC,CEB,RQD3,data                 .
                                   o-------------------------->
                                                               WHILE rc=OK and
                                                                     (wr=DATA_COMPLETE or
                                                                     wr=DATA_INCOMPLETE)
                                                                    .
                                                                   rc=OK, wr=CONFIRM_DEALLOCATE
                                                                    .
                                                               MC_CONFIRMED ...
                                   +RSP                             .
                                   <--------------------------o     .
                                                                    .
    rc=OK                                                           .
        .                                                           .
```

Figure 3. File Transfer (Batch Send)

transmission speeds. The client and server simply send and receive records; they do not have to worry about record boundaries.

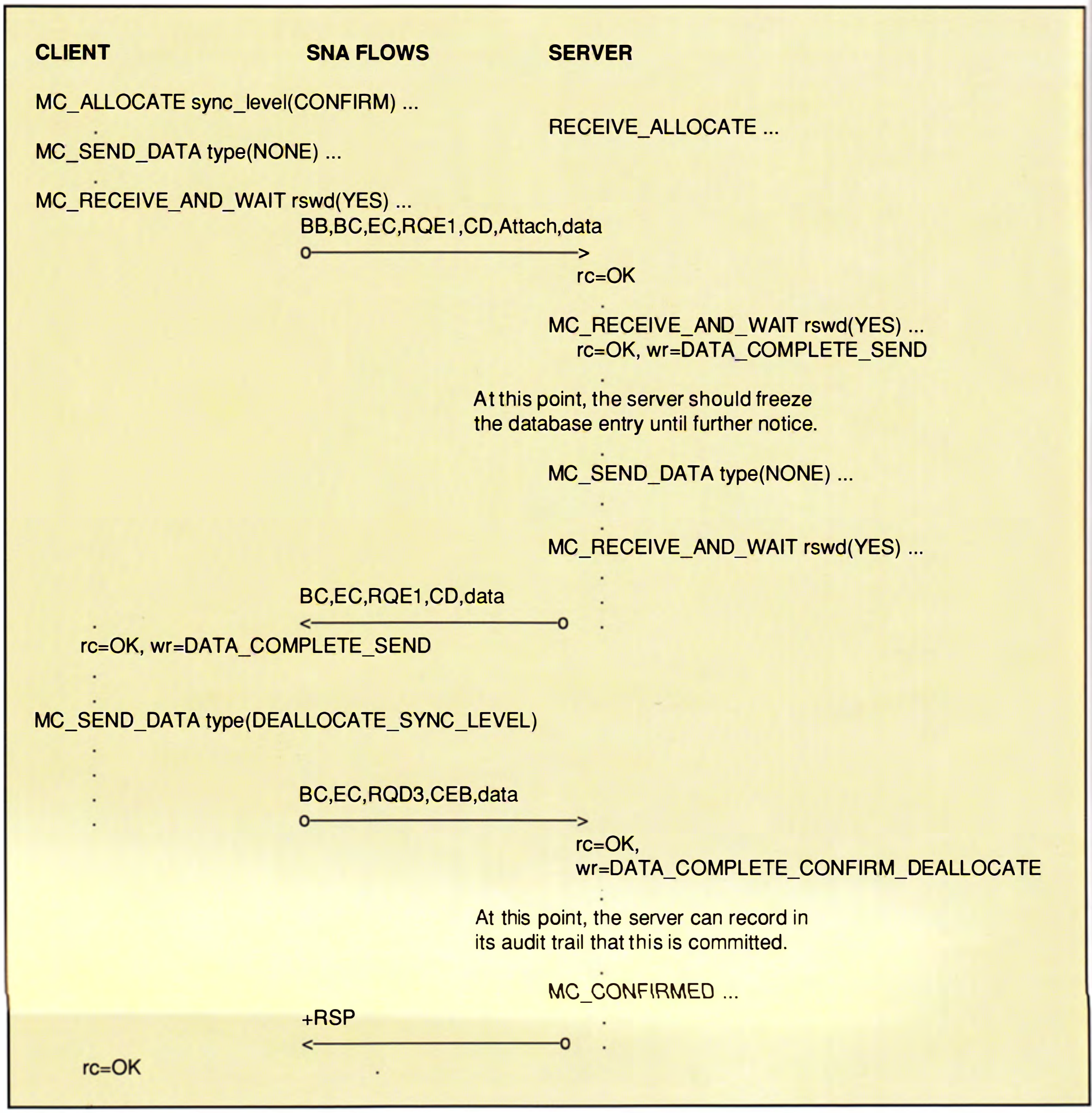
The file transfer transaction is typically used to send relatively large amounts of data from one location to another. It is used in situations where several related data records need to be moved. For example, SNA Distribution System (SNA/DS) implementations such as DISOSS use this transaction to distribute files. It is also called the *batch send* transaction because a batch of records is sent at the same time.

In this transaction, the client uses one or more SEND_DATA verbs to transfer the contents of a file. The server receives data with a loop of RECEIVE_AND_WAIT verbs until it is clear that there is no more data to receive.

In Figure 3 you will find pseudocode that implements this transaction using mapped conversation verbs.

THE DATABASE UPDATE TRANSACTION (CONVERSATIONAL REPLY)

The database update transaction is typically used to update information managed by a partner program. It is used in database applications where a client program requests



```
CLIENT                          SNA FLOWS                      SERVER

MC_ALLOCATE sync_level(CONFIRM) ...
      .                                                        RECEIVE_ALLOCATE ...
MC_SEND_DATA type(NONE) ...
      .
MC_RECEIVE_AND_WAIT rswd(YES) ...
                                BB,BC,EC,RQE1,CD,Attach,data
                                o-------------------------->
                                                                  rc=OK
                                                                   .
                                                               MC_RECEIVE_AND_WAIT rswd(YES) ...
                                                                  rc=OK, wr=DATA_COMPLETE_SEND
                                                                   .
                                                     At this point, the server should freeze
                                                     the database entry until further notice.
                                                                   .
                                                               MC_SEND_DATA type(NONE) ...
                                                                   .
                                                                   .
                                                               MC_RECEIVE_AND_WAIT rswd(YES) ...
                                                                   .
                                BC,EC,RQE1,CD,data                 .
      .                         <-----------------------o          .
   rc=OK, wr=DATA_COMPLETE_SEND
      .
      .
MC_SEND_DATA type(DEALLOCATE_SYNC_LEVEL)
      .
      .
      .                         BC,EC,RQD3,CEB,data
      .                         o-------------------------->
                                                                  rc=OK,
                                                                  wr=DATA_COMPLETE_CONFIRM_DEALLOCATE
                                                                   .
                                                     At this point, the server can record in
                                                     its audit trail that this is committed.
                                                                   .
                                                               MC_CONFIRMED ...
                                +RSP                               .
                                <-----------------------o          .
   rc=OK                                   .
```

Figure 4. Database Update (Conversational Reply).

information from the database server, updates the information, and returns the update to the server to be set into the database.

This transaction is really an inquiry transaction followed by a credit check transaction performed on the same conversation. A single conversation is necessary to control the scope of recovery. It shows the transfer of responsibility between the client and the server.

This same transaction may be used when a client requests information from a server which in turn requests information (usually for accounting or auditing purposes) from the client. In this case, the server appends the new request to the reply sent to the original request, and the result is returned in another record. In this environment, the transaction is called a *conversational reply* because the reply solicits an additional reply from the originator.

In Figure 4 you will find pseudocode that implements this transaction using mapped conversation verbs.

THE PIPELINE TRANSACTION (ONE WAY BRACKET)

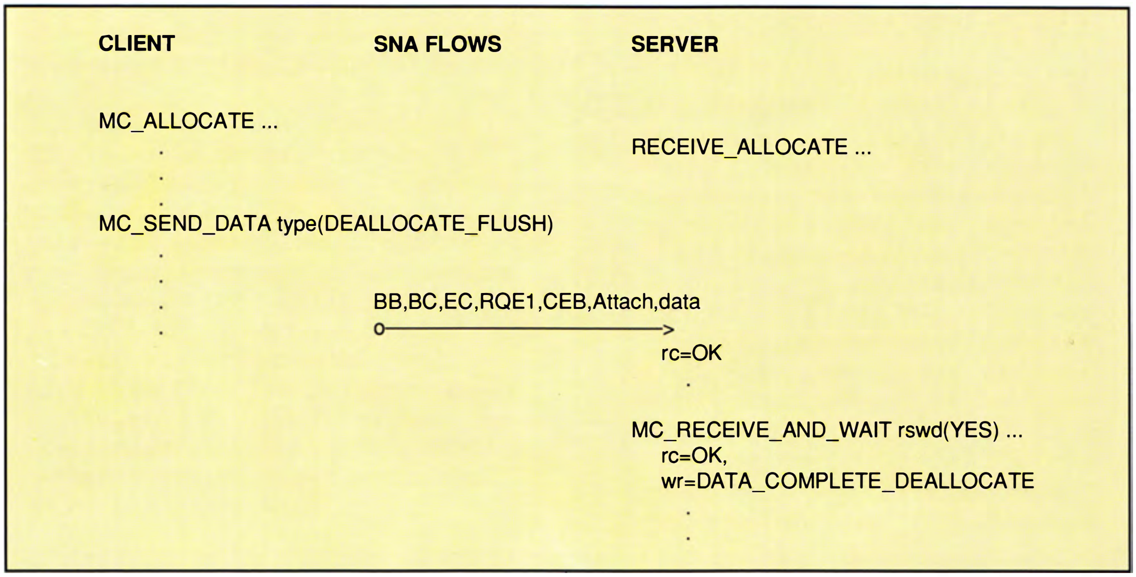
This transaction is the simplest to show, yet often the most difficult to explain. It is generally used when the client or server is severely limited in its resources. The client sends a chunk of data and then leaves. There is no positive acknowledgement of the data.

Applications must correlate their requests and replies, because they are handled as independent conversations and there is no guarantee that the server completely processed an incoming record. A pipeline transaction is used in situations where the application complexity and APPC resource trade-off is made in a way to minimize the APPC resources.

The pipeline transaction is typically used to pipe data between applications. For example, Advanced Peer-to-Peer Networking (APPN) uses this type of transaction to transmit topology updates and to implement resource searches.

This transaction is often used when a large number of independent units of work need to be done but it is impossible to allocate enough conversations and sessions so that they can run in parallel. That is, the arrival rate is too large to be handled by the network resources or the server. This can occur when communicating to CICS where VTAM does not support parallel sessions from a node outside the subarea. It can also occur when a node contains insufficient storage or when an impossibly large number of transactions must be run in parallel.

In this situation, you will write an application that uses two conversations: one



```
CLIENT                    SNA FLOWS                SERVER


MC_ALLOCATE ...
     .                                             RECEIVE_ALLOCATE ...
     .
     .
MC_SEND_DATA type(DEALLOCATE_FLUSH)
     .
     .
     .                    BB,BC,EC,RQE1,CEB,Attach,data
     .                    o------------------------>
                                                     rc=OK
                                                     .

                                                   MC_RECEIVE_AND_WAIT rswd(YES) ...
                                                     rc=OK,
                                                     wr=DATA_COMPLETE_DEALLOCATE
                                                     .
                                                     .
```

Figure 5. Pipeline (One-Way Bracket).

to send data and one to receive. The application will build a correlator into each data record sent to identify the unit of work it belongs to. The partner will return the correlator in related data records so that the application can correlate the data to the correct unit of work. Effectively, the application is building conversations on top of the APPC conversation pair.

This transaction is known as the *one-way bracket* because the SNA line flow bracket begins and ends without generating any return flow. It is also known as a *datagram* in some systems.

In Figure 5 you will find pseudocode that implements this transaction using mapped conversation verbs.

Editor's note: Look for more about IBM's SAA Networking Services/2 in the Summer 1991 of the "IBM Personal Systems Developer."

ACKNOWLEDGMENTS

We appreciate the careful review and many helpful comments offered by a number of our IBM associates (listed below alphabetically).

Jim Gray, Rick McGee, Paul McKenna, Peter Schwaller, Bill Siddall, and Mike Wamboldt.

REFERENCES

IBM SAA Networking Services/2 APPC Programming Reference. IBM Document number SC52-1112.

IBM Systems Network Architecture Transaction Programmer's Reference Manual for LU Type 6.2. IBM Document number GC30-3084.

Alex Berson. *APPC: Introduction to LU 6.2,* McGraw-Hill Publishing Co., New York, 1990. ISBN 0-07-005075-9.

Lance D. Bader, *IBM Corporation, E42/673, P.O. Box 12195, Research Triangle Park, NC 27709, (919) 254-4461. Mr. Bader, currently a Senior Programmer in APPC Market Enablement, joined IBM in 1977 to develop data communication device drivers for the IBM Series/1 Realtime Programming System. Since then, he has participated in developing architectures and products for a wide variety of data communication protocol stacks. He has a BS in Computer Science from North Dakota State University.*

John Q. Walker II, *IBM Corporation, E42/673, P.O. Box 12195, Research Triangle Park, NC 27709, (919) 254-4414. Mr. Walker manages one of the development teams responsible for the implementation of APPC and APPN for OS/2, IBM SAA Networking Services/2. He joined IBM in Rochester, Minnesota in 1978, where he was involved with the development and testing of the operating-system software for the IBM System/38. In Research Triangle Park, John was an architect for the IBM Token-Ring Network, serving as a co-editor of IEEE 802.5 local area network standards, 1983-1984. He received a BS, BA, and MS from Southern Illinois University and is completing a Ph.D. in computer science at the University of North Carolina at Chapel Hill.*

Presentation Manager

Client-Server Computing Using PM's Dynamic Data Exchange

by Michael R. MacFaden

Michael R. MacFaden

Client-Server architecture defines the computing software platform for the 1990s. Presentation Manager's Dynamic Data Exchange (DDE) protocol implements client-server architecture in an efficient and flexible manner that is sure to change the nature of how people do their computing. With OS/2's ability to let users run multiple applications at once, the need for those applications to work together is all the more important. This article discusses the need for DDE and demonstrates the basics of programming with DDE.

In the past, tying data from different applications together was usually limited to one application reading or translating another application's data file. Getting a table of numbers from a spreadsheet to a word processor was a major chore. Every time the data was modified in the spreadsheet, it had to be manually transferred back to the document. Selecting new software packages meant checking for compatible file support of the few available formats such as DIF, CSV, and CGM.

The first real solutions for solving the general task of combining data from different sources were software products marketed in the middle 1980s such as Lotus® Jazz™ and Ashton-Tate's Framework®. On mainframe computers, IBM introduced a large software product called Application System® (AS). In the mini-computer arena, UNIX systems embodied this capability early in the 1970s through inter-process communications (IPCs) called pipes. Simple, one-function programs that read from and write to a pipe can be seen even in the DOS COMMAND.COM shell. A DOS example is:

```
dir | sort /+17 | more
```

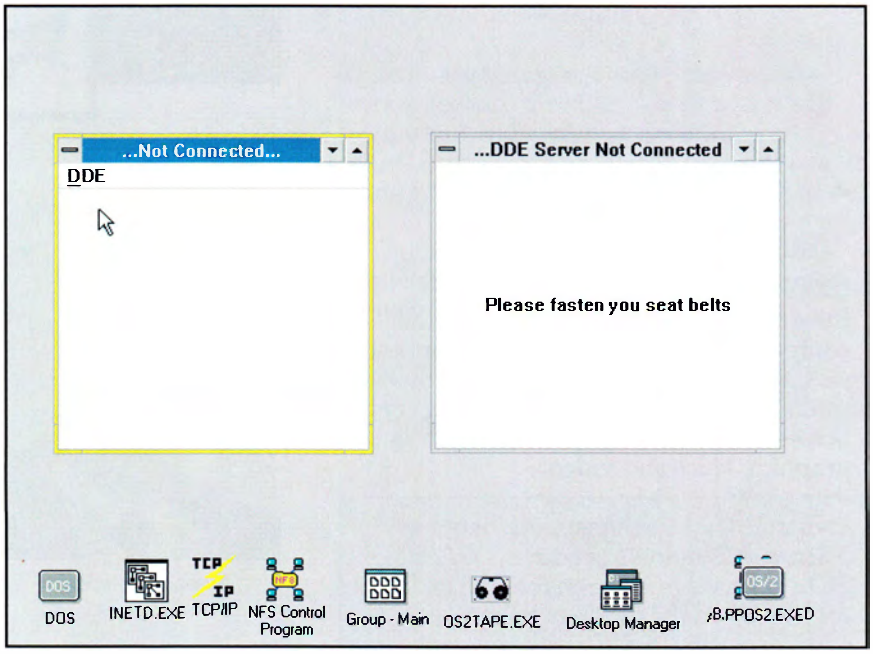
Users liked one component of these integrated solutions, but found another component lacking. Today customers still seek to create seamless work flows that span multiple applications from different vendors across a variety of data formats such as text, graphics, voice and video.

Dynamic Data Exchange originated in Microsoft Windows product. OS/2 PM's DDE is very similar. See *Franklin and Peters* for details on the differences between Windows and OS/2 DDE. Any application that implements DDE can negotiate and exchange data with any other DDE application. Customers should select every component in their work flow and can feel confident that not only will each application have a similar interface, but each component application will work seamlessly with the other. Richard Hale Shaw writes in the May 1990 issue of the *Microsoft Systems Journal*, (p. 81):

> DDE can be considered as a bridge between diverse applications: by providing a uniform means of transferring data between otherwise dissimilar applications, it offers applications a level of integration that has been long talked about but never, until now, achieved.

DDE can be considered as a bridge between diverse applications.

Now people can build a financial model in Microsoft Excel, see how it affects product design in AutoCad®, and render executive proposals in DeScribe® V2.0 Word Publisher. By implementing DDE in their application, developers enable people to seamlessly tie together undreamed-of combinations of applications. Using DDE, programmers can benefit by focusing on one component exclusively, and rely on other products to take care of critical but mundane componentry such as word processing, graphics, printing, or database access.



Please fasten you seat belts

Figure 1. Initial Screens from Sample Programs

A note of caution is in order: DDE is still evolving. Many areas of the protocol are not defined in either IBM or Microsoft OS/2 manuals. These areas include how to maintain a DDE hot link across system boots, and support for multiple hot links over a single conversation.

Microsoft, in conjunction with Lotus Development Corp., Aldus Corp., and Word Perfect Corp. has defined extensions to the DDE protocol to handle these cases and more. These extensions, currently called *Object Linking and Embedding* (OLE), are available on the Microsoft Public Access BBS listed on the back page of most *Microsoft System Journal* issues.

The rest of this article looks at the current PM DDE APIs, and describes a sample program available to IBMLink subscribers on the ESDTOOLS disk. By starting with this small example, the reader will understand the basic mechanics of DDE and be better prepared for OLE.

DDE API

Readers should be familiar with basic Presentation Manager based programs, C Language pointers and structures, and OS/2 shared memory. The OS/2 DDE API set is covered in the following OS/2 Version 1.2 manuals:

- *Microsoft OS/2 Programmer's Reference*, Vol. 1 Chapter 27

- *IBM OS/2 Programming Guide*, Chapter 29

THE SAMPLE PROGRAMS

The programming examples used in this article are called Client.exe and Server.exe. The server program can service a single client at a time and provide a quotation on demand. The server displays its current quote in its window, updating it every five seconds. The client, once it establishes a conversation with a server, can request a copy of the current quotation or ask to have the server send a new copy every time the quotation changes. The client program has a menu bar which allows the user to execute DDE transactions. Figure 1 shows the two programs as they first appear on the screen.

In a typical scenario, both the client and server are brought up, then the user selects the *Start Conversation...* menu item from the client, and issues individual requests or a hot link for quotations. The items in the menu will enable and disable themselves based on the state of the conversation which is reflected in both the client and server's title bars. Multiple instances of either program may be invoked at the same time.

As with all client-server applications, development starts with the most common design and coding tasks such as:

- Who is the Client?
- Who is the Server?
- Initiating a conversation
- Exchanging data
- Terminating a conversation

We will explore each as they relate to DDE.

WHO IS THE CLIENT? WHO IS THE SERVER?

Some applications can be both a DDE client and server at the same time. For example both Lotus 1-2-3/G and Microsoft Excel on the same system, select a range of cells and issue an Edit->Copy on the first spreadsheet, then an Edit->Paste Link on the other. Reverse the process so that Excel hot links a group of cells in a spreadsheet to a Lotus spreadsheet. This allows the user to update data in one application on an on-going basis and see the changes reflected in another.

Examples of single purpose or pure server programs include applications that read thermometers wired to the serial port, that process incoming stock transactions over an FM channel, or that attach a transaction program to query a remote database.

The programming examples described below do not cover every aspect of the DDE protocol. One major DDE item not covered in this article is how to implement command processing such that the client can write or issue macros against a server. Specifically the example programs do not show the window messages WM_DDE_EXECUTE and WM_DDE_POKE.

While the sample DDE server can only handle a single client, multi-client support is the norm. DDE programs can handle an unlimited number of clients. Server.exe *supports data exchange of text strings* — the single default data format — with client applications. Again, DDE does not limit what kind of data can be exchanged. See *Franklin and Peters* listed in the bibliography for a more complex sample of DDE programming that shows how to handle multiple clients and application defined data types.

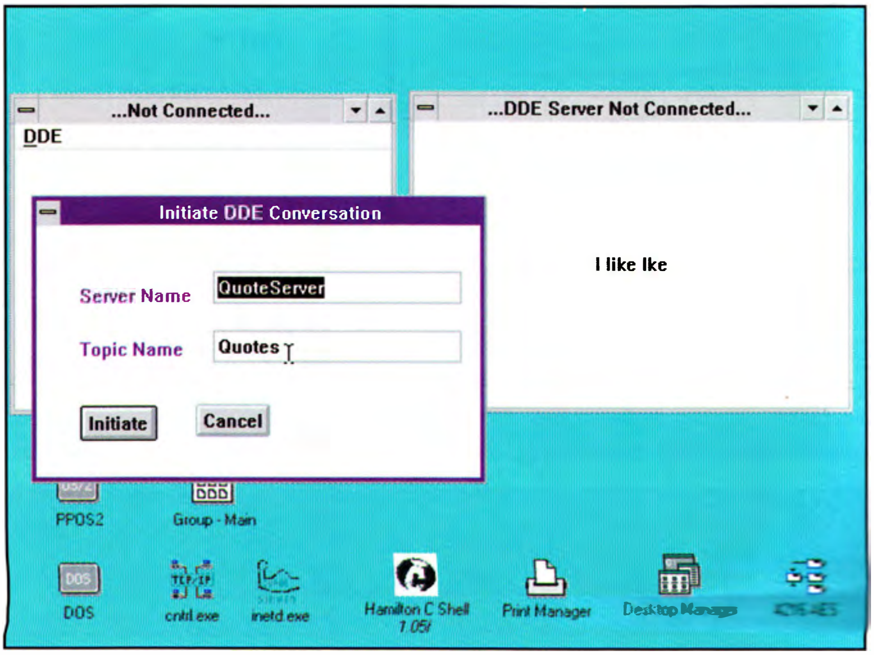
INITIATING A CONVERSATION

Typically a server application is invoked before the client. Both must be running before the client issues a WM_DDE_INITIATE window message.

The transaction to start a conversation, assuming both client and server are running, looks like this:

Client.exe Sends
 WinDdeInitiate(application, topic)
Server.exe Responds
 WinDdeRespond(hwndClient,application,topic)
The conversation is now established.

Figure 2 shows a dialog box in which the user can enter an application name and topic to send out to all frame windows. The first valid reply (WM_DDE_INITIATEACK) will begin the conversation.



Server Name QuoteServer

Topic Name Quotes

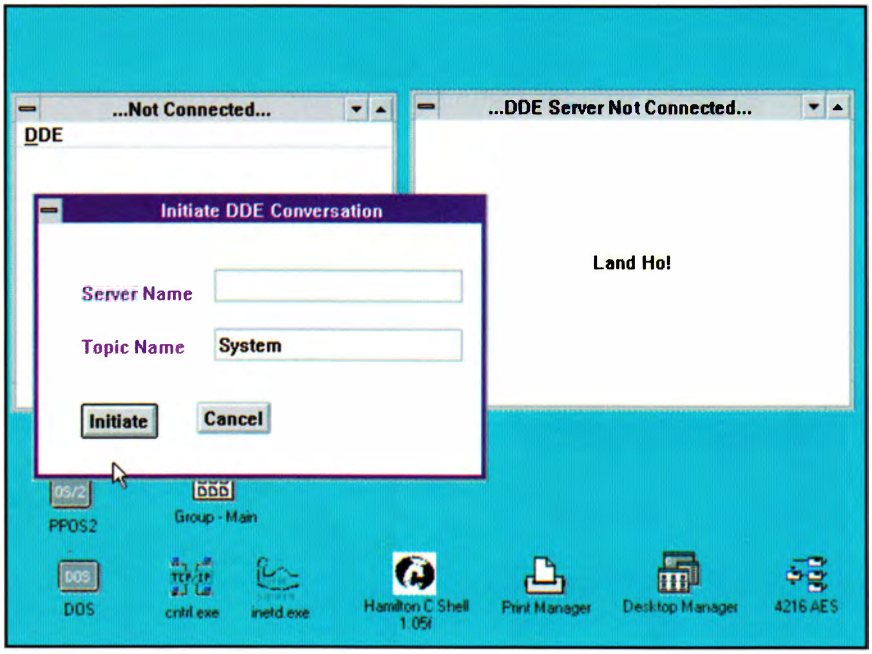
I like Ike

Figure 2. Starting a DDE Conversation

Since multiple acknowledgments might be received after broadcasting a WM _DDE_INITIATE call, the client has to choose which one it wants and release the others by sending a WM_DD_TERMINATE message to them. A negotiation is nothing more than two applications agreeing what to converse about. In DDE terms, this means agreeing primarily upon a topic.

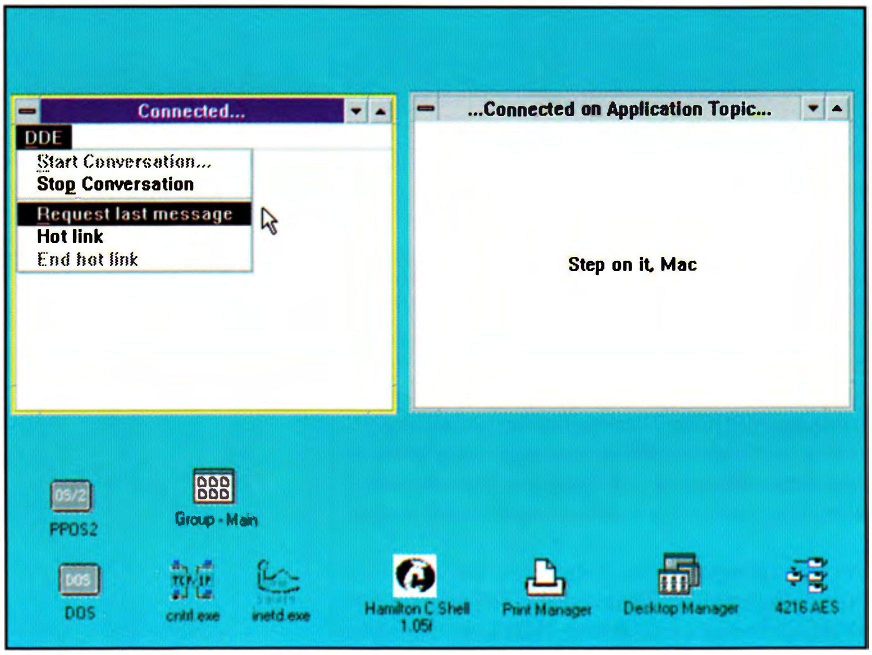
A client program can generate a list of currently available servers by issuing a WinDdeInitiate call using **NULL** as the application and **System** as the topic. Then the client can query a particular server about topics and data formats supported.

An alternative, described by Richard Hale Shaw (*Microsoft Systems Journal, Vol. 5, No. 3, p. 84, May 1990*), allows the client to have a server enumerate all data formats by sending a NULL Topic in the WinDdeIniate call.



Land Ho!

Figure 3. Making the Client Program Broadcast on the SYSTEM Topic



Step on it, Mac

Figure 4. Requesting Data from the DDE Server

Programmers should check with the product supplier to see which method is used.

The application can have as many open conversations and topics between the client and server as the server can handle. The example server supports the System topic and an application defined GET-A-QUOTE topic. According to Microsoft and IBM manuals, all applications should support the System topic. The System topic is described in the Microsoft *Programmers' Reference* Volume 1, p. 354 or the IBM *Programming Guide*, Chapter 29, p. 10. Figure 3 shows how to make the client program broadcast on the System topic.

The sample Client.exe program externalizes DDE API transactions. This allows programmers to get a feel for how DDE works. A real application would hide these functions either through a Drag & Drop Direct Manipulation type interface or through a Cut/Paste Link interface. Other interfaces are up to the programmer's imagination.

EXCHANGING DATA

Data exchange primarily runs as follows: the client issues requests and the server returns the data. Unsolicited data can be sent back to a server using the WM_DDE_POKE message. Data is only exchanged after a client and server have established a conversation. A client can have parallel conversations if the server supports multiple conversations.

The typical single data request transaction is:

```
Client.exe Sends
 WinDdePostMsg(hwndServer,WM_DDE_REQUEST...)
Server.exe Responds
 WinDdePostMsgChwndClient,WM_DDE_DATA...)
```

Figure 4 shows how to make the client process request the current quotation from the server.

A hot link type of data transaction looks like the following:

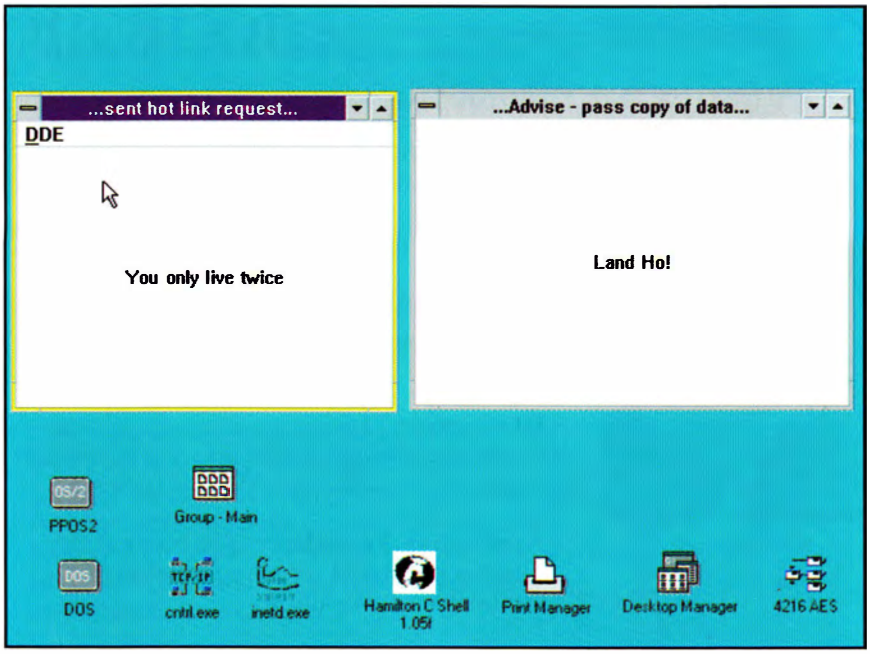
```
Client.exe Sends
 WinDdePostMsg(hwndClient, WM_DDE_ADVISE...)
Server.exe Responds
 WinDdePostMsg(hwndClient, WM_DDE_DATA...)
Server.exe Responds
 WinDdePostMsg(hwndClient, WM_DDE_DATA...)
Server.exe Responds
 WinDdePostMsg(hwndClient, WM_DDE_DATA...)
Client.exe Sends
 WinDdePostMsg(hwndServer,WM_DDE_UNADVISE..)
```

Figure 5 shows the client process hot linked to the server. The client can then issue one-time data requests or ask the server to advise it of any changes to a particular topic/item combination as shown above. This second method of data exchange demonstrates the true power of this protocol. It allows a single piece of data to be represented in many formats (graphical, text, spreadsheet); and in many different applications at once such as a graphics package, word processor, spreadsheet, database, or personal information manager).

DDE usage should not be ruled out if the data to be exchanged is larger than a single OS/2 memory segment (64KB). Passing huge pointers, handles to files or notices that the data has changed are appropriate ways to handle these larger data segments.

If data is sent in multiple passes, or if it is necessary for one side to back down because it is busy or can't handle a request, the DDESTRUCT structure's *fs* status field should be used. This status field can be set to make all transactions require acknowledgements and provide additional error information. If this DDESTRUCT isn't big enough to handle the error requirements, an additional conversation on the System topic and ReturnMessage item can be used to add additional application-specific data.

Since DDE currently only defines one format, DDEFMT_TEXT (according to Microsoft manuals and CF_TEXT in IBM manuals), applications wishing to exchange data in non-text formats need a way of registering the data types to ensure uniqueness. *Franklin and Peters* suggest applications can issue calls to the Atom Table described in the IBM *Programming Guide*, chap. 13 and in the Microsoft *Programmers Reference* Vol. 1, chap. 55. OLE extends DDE formats to include Clipboard formats, ObjectLink, OwnerLink, Native and Binary.



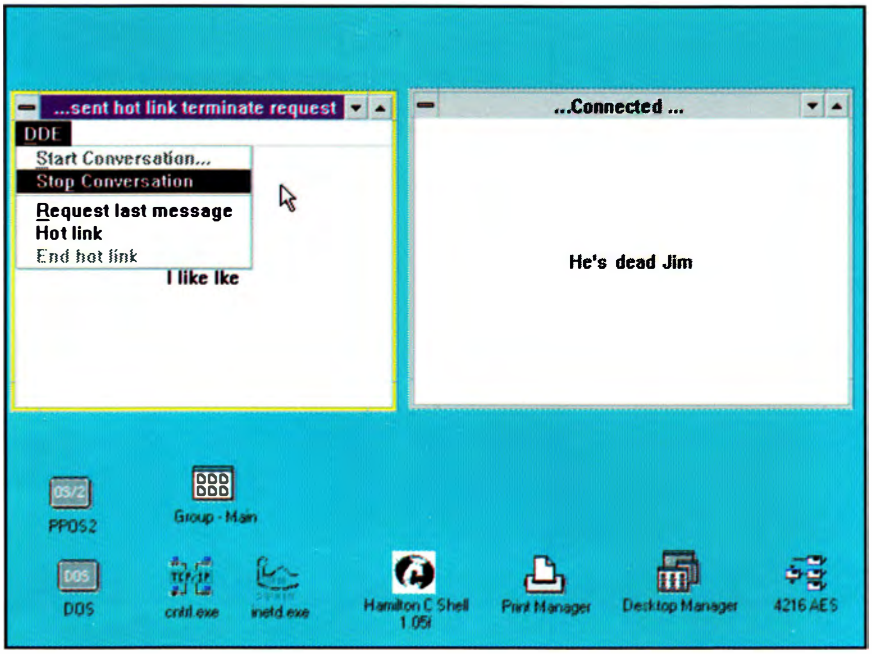
...sent hot link request...

You only live twice

...Advise - pass copy of data...

Land Ho!

Figure 5. Requesting a Hot Link from the DDE Server



...sent hot link terminate request

I like Ike

...Connected ...

He's dead Jim

Figure 6. Terminating a DDE Conversation

TERMINATING A CONVERSATION

Either side can terminate the conversation at any time. Be sure both applications trap the WM_QUIT and WM_CLOSE messages to see if a DDE conversation is in progress, in order to shutdown the conversation correctly.

```
Client.exe Sends
  WinDdePostMsg(hwndServer,WM_DDE_TERMINATE...)
Server.exe Responds
  WinDdePostMsg(hwndClient,WM_DDE_TERMINATE...)
```

Figure 6 shows the client terminating the conversation.

GLOSSARY

DDE Client: a program that initiates a conversation with a server with a request for data.

DDE Server: a program that responds to client requests and enters into a conversation based on mutually agreed upon topics.

Application/Topic/Item: The method by which a client and server negotiate to establish a conversation.

Conversation: A state in which two PM applications can exchange data. A Conversation is managed by the applications themselves based upon the Application/ Topic pair and subsequent negotiations.

Hot Link: Sometimes called link or advise. A state in which a server is directed to keep the client informed about a selected topic and item.

REFERENCES

Franklin, Susan and Peters, Tony. "A Technical Study of Dynamic Data Exchange Under OS/2." *Microsoft Systems Journal*. Vol. 4 No. 3. May 1989: 1-16p.

IBM *OS/2 Programming Tools and Information Version 1.2 Presentation Manager Programming Reference*, Volumes 1 , 2. IBM Corp., 1989.

IBM *OS/2 Programming Tools and Information Version 1.2 Programming Guide*, 1989, Chapter 29.

Microsoft Corp. *Object Linking and Embedding*, (January 1991, beta version).

Petzold, Charles. *Programming the OS/2 Presentation Manager*, (Chaps 10, 13, 14, Microsoft Press, 1989).

Shaw, Richard Hale, "Implementing DDE with Presentation Manager Object Windows." *Microsoft Systems Journal* Volume 5 No. 3. May 1990: 81-88p.

Shaw, Richard Hale, "Providing DDE Facilities to OS/2 Kernel Applications," *Microsoft Systems Journal*, Volume 5 No. 4. July 1990: 65-84p.

Michael R. MacFaden, *IBM Marketing & Service Division. IBM U.S., 1501 California Avenue MS #13. Palo Alto CA 94304 (Internet E-mail address: macfaden@paloic1.vnet.ibm.com), joined IBM in 1988 as an applications programmer for VM/370 systems. He is currently an OS/2 applications programmer in the Workstation Application Environment Department. Mr. MacFaden holds a BS in Business and Management Information Systems from California Polytechnic State University at San Luis Obispo.*

Presentation Manager

Client-Server Architecture for Single-User Applications

by Randell S. Flint

The standard model of client-server software divides any particular application into two distinct programs: a server which manages data (or other shared resources) and a client which provides user access to that data. The client and server communicate with each other, the client asking the server to perform action on its behalf. Together, the client and server cooperate to accomplish the tasks which make up the complete application.

Typically, multiple copies of the client can communicate with the server. Thus several users can share access to the same information. As a result, client-server computing is most often thought of as an architecture for systems involving multiple users.

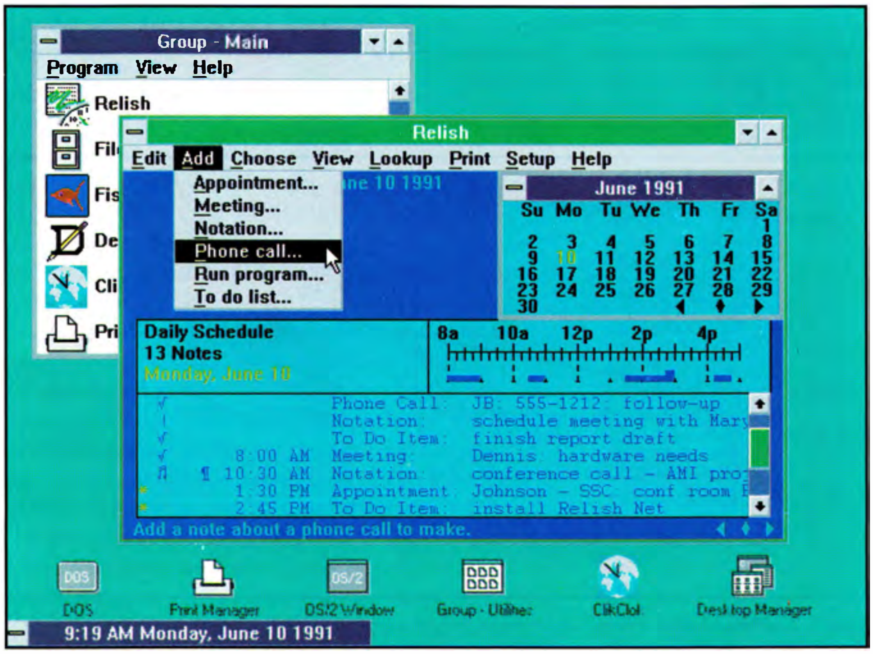
But client-server architecture isn't limited to multi-user applications. Traditional, single-user applications can also be designed and implemented using client-server architecture. This applies to both custom applications, designed for in-house corporate use, as well as common software tools such as word processors, accounting programs, spreadsheets, and databases. Using the client-server approach, even in single-user applications, can provide significant benefits not only during the original design and implementation, but also during the inevitable evolution of the application over time.

This article considers how client-server architecture is applied to single-user applications. In so doing, it examines how client-server architecture was used in designing Relish™, the desktop time and information manager for OS/2 Presentation Manager from Sundial Systems Corporation (see Figure 1).

CLIENTS AND SERVERS DON'T HAVE TO BE ON SEPARATE SYSTEMS

Dropping down below the level of the application and its functions, the standard view of client-server computing envisions two (or more) systems connected on a LAN. One system acts as the server and the other(s) as client(s). In an order processing application, for example,

Randell S. Flint



Daily Schedule
13 Notes
Monday, June 10

```
 √                  Phone Call:   JB: 555-1212: follow-up
 !                  Notation:     schedule meeting with Mary
 √                  To Do Item:   finish report draft
 √        8:00 AM   Meeting:      Dennis: hardware needs
 ♫    ¶  10:30 AM   Notation:     conference call - AMI proj
*         1:30 PM   Appointment:  Johnson - SSC: conf room B
*         2:45 PM   To Do Item:   install Relish Net
```

Add a note about a phone call to make.

9:19 AM Monday, June 10 1991

Figure 1. Relish Time and Information Organizer On The Desktop

the order database is managed on the server while order entry takes place on multiple clients (see Figure 2).

In reality, this LAN-based hardware topology is largely irrelevant to the more basic issues of client-server computing. The notion of client and server *systems* misses the point; the systems aren't communicating, per se. What is really going on is communication between client and server *programs* — one is running on a workstation system and the other is running on a network server system. In other words, copies of the order entry *program* communicate with the order database *program* (see Figure 3).

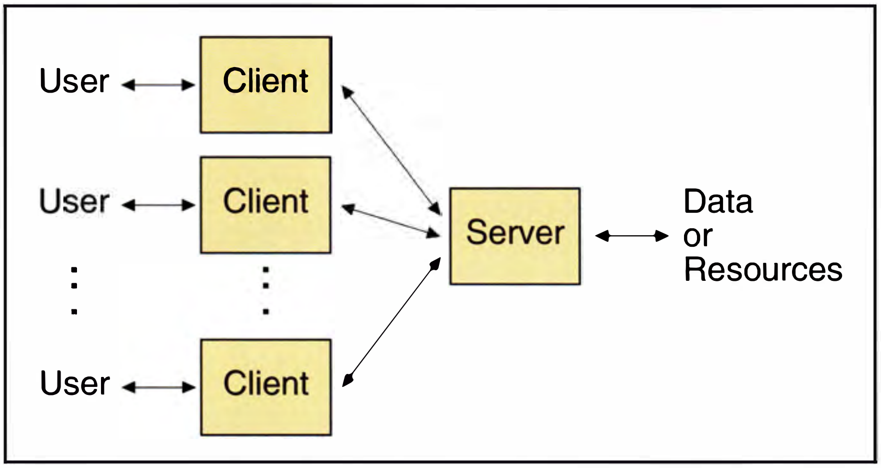


Figure 2. The Logical View of Client-Server Architecture

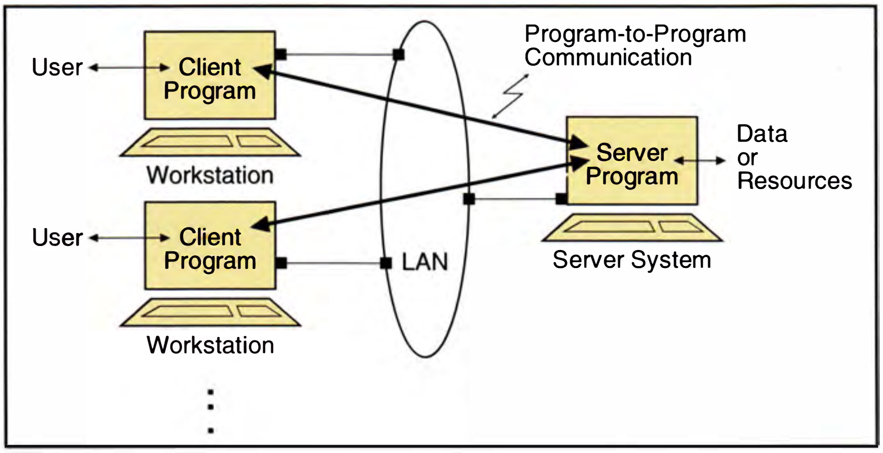


Figure 3. The Physical View of Client-Server Architecture on a LAN

In a LAN-based client-server system, it just happens that each client and server program runs on a separate system. While this is a common topology, it is not the only one possible. Two programs running on two workstations could be involved in a client-server relationship, one providing *services* and the other using them. In fact, it doesn't even take separate systems to do client-server computing. A client might run on the same machine as the server — or two clients might run on the same workstation, but share access to a server.

All client-server computing requires is two (or more) programs willing and able to talk to each other, an environment that allows them to run simultaneously, and some mechanism for them to communicate with each other.

OS/2 provides just such an environment. Multitasking is just what it takes for two programs to run together. Interprocess communication enables them to talk to each other. As another author has put it, "One of the most powerful tools under OS/2 for creating applications is the ability to create multiple processes that communicate with each other." [1]

OS/2 provides a rich variety of interprocess communication techniques to connect one task with another. This article won't attempt to detail how and when these various schemes can or should be used. Briefly, however, they include named pipes, (ordinary) pipes, queues, shared memory, semaphores, and signals. Under Presentation Manager, window messages, Dynamic Data Exchange (DDE), and Object Linking and Embedding (OLE) provide other interprocess communication options.

It is important to note that this rich variety is available in ordinary OS/2; still more options are available in the LAN Server/Requester or Microsoft LAN Manager environments that are the focus of other articles in this issue. This article concentrates on what is possible in single-user OS/2, regardless of whether it is connected to a local area network or not.

[1] Ryan, Ralph. *The Microsoft LAN Manger: A Programmer's Guide*. Microsoft Press, 1990.

CLIENT-SERVER ARCHITECTURE IN SINGLE-USER APPLICATIONS

At this point you may be asking, why use client-server computing in a single-user system... Isn't the whole point of client-server computing to share information (or other resources) among different users? The answer is both yes and no. Certainly there is great benefit to sharing information and resources among users, but many of these same benefits also apply to single-user systems.

Traditionally, system design has been based on the thought that a user can do only one thing at a time. That's true for most mainframe environments as well as traditional DOS-based PCs. Most often this is because of limitations in the underlying environment; it's hard to let the user do two things at once even if that's what he/she wants to do.

But OS/2, and particularly the Presentation Manager desktop, have changed what a user can and does do. Multiple windows (to one or more applications) allow the user to easily switch back and forth between one activity and another. In fact, user-centered computing (where the user, not the program, is in primary control of the interaction between human and computer) *demands* that the user be able to move easily from one activity to another and back.

For the program designer, this can mean significantly extra effort. Those multiple windows may need to share information (or other resources); they may be associated with the same or different programs. Coordination among them may be extremely important and difficult to implement with a single program since, with today's programming tools, it's still easiest to write programs that deal with only one thing at a time.

That's where client-server computing can come to the rescue: two distinct programs (client and server) in place of one that is trying to do *all* the work. For instance, consider a catalog shopping application on the Presentation Manager desktop. It's natural to think of it in client-server terms: the windows displaying product information are each managed by a client program; the catalog data they share is managed by a server program. This client-server division is just as true when the windows and the data are on one system as it is when product descriptions are being viewed on several distinct workstations and the catalog is managed on a network server.

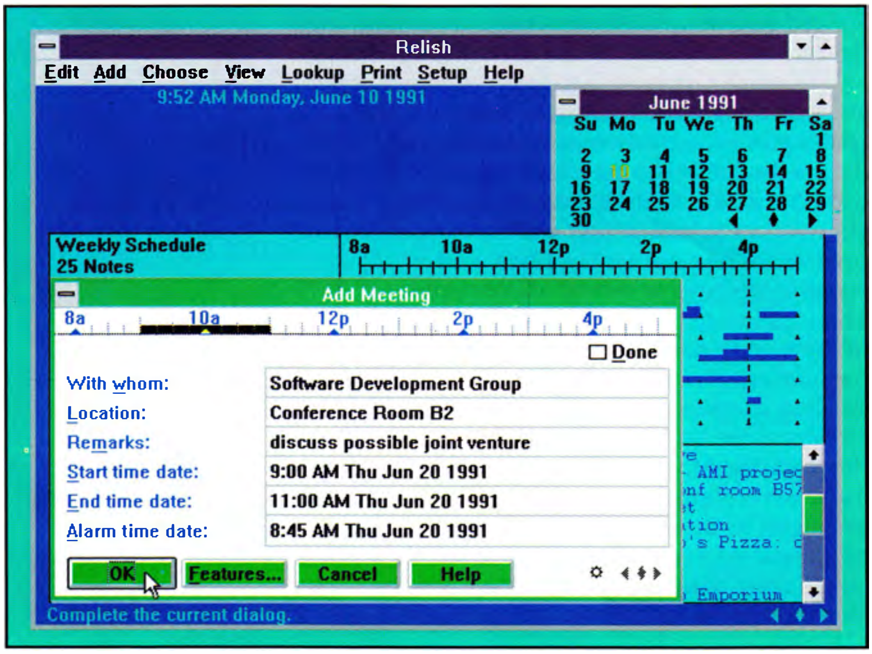
Or consider a word processor with its spelling checker. When an error is found, the spelling checker may well carry out a complex dialog with the user concerning alternatives to misspelled words, various correction/replacement options, and possible updates to private dictionaries. These programs aren't sharing information in the traditional sense of client-server computing. But it is natural to think of the spelling checker as providing *correction services* to the word processor. On the Presentation Manager desktop, this could mean several open documents in windows managed by one or more copies of the word processor interacting on demand with a single copy of the spelling checker and its *shared* dictionaries.

Single-user applications can be implemented with client-server architecture.

RELISH FOR OS/2 PRESENTATION MANAGER

Relish, the desktop time and information organizer from Sundial Systems, is an example of client-server architecture applied from the ground up in the design and implementation of a single-user application. It is designed to provide the user with calendar, reminder, and personal scheduling capability in an intuitive, reliable, and easy-to-use way. Enhancing the features found in most desktop calendar programs with the advanced technology of OS/2 Presentation Manager, Relish provides unmatched flexibility coordinating tasks and managing time. For example, a small monthly calendar can remain on the desktop when the program is minimized, offering both immediate access to any day's schedule and the ability to schedule commitments without having to restore the main window and search for the correct day.

Conceptually, the Relish user creates and manipulates *notes*. Each note corresponds to one entry on his/her schedule and/or To Do list. The dialog used to create a new note is shown in Figure 4. Whenever a schedule is displayed, a one-line summary of the note is generated and displayed.



9:52 AM Monday, June 10 1991

Weekly Schedule
25 Notes

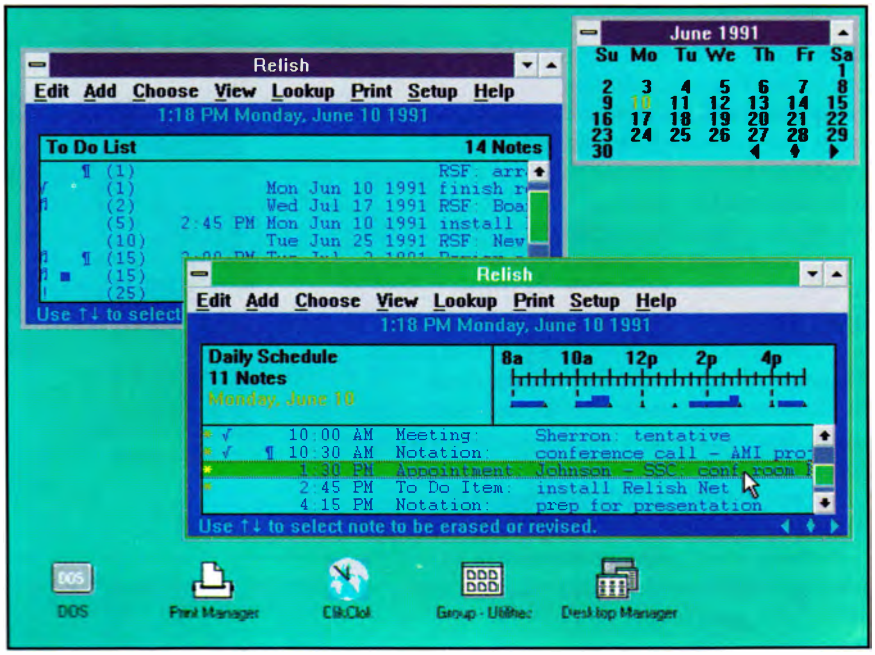
☐ Done

With whom:	Software Development Group
Location:	Conference Room B2
Remarks:	discuss possible joint venture
Start time date:	9:00 AM Thu Jun 20 1991
End time date:	11:00 AM Thu Jun 20 1991
Alarm time date:	8:45 AM Thu Jun 20 1991

```
e
- AMI projec
onf room B57
et
ation
o's Pizza: c
o Emporium
```

Complete the current dialog.

Figure 4. Adding a Note About a Meeting



1:18 PM Monday, June 10 1991

To Do List 14 Notes

```
     ¶ (1)                         RSF: arr
√      (1)          Mon Jun 10 1991 finish r
¶      (2)          Wed Jul 17 1991 RSF: Boa
       (5)  2:45 PM Mon Jun 10 1991 install
       (10)         Tue Jun 25 1991 RSF: New
¶    ¶ (15) 2:00 PM Tue Jul  2 1991 Review 
¶    ■ (15)
       (25)
```

Use ↑↓ to select

1:18 PM Monday, June 10 1991

Daily Schedule
11 Notes
Monday, June 10

```
* √    10:00 AM  Meeting:      Sherron: tentative
* √  ¶ 10:30 AM  Notation:     conference call - AMI pro
*       1:30 PM  Appointment:  Johnson - SSC: conf room B
*       2:45 PM  To Do Item:   install Relish Net
        4:15 PM  Notation:     prep for presentation
```

Use ↑↓ to select note to be erased or revised.

Figure 5. Daily Schedule and To Do List in Two Relish Windows

The notes are stored in a database managed by Relish. It is important to realize that, from the user's perspective, Relish is not file oriented. The user does not open a file of appointments, manipulate parts of it, and then close it. The intent is that the user not care (or know) where (or how) his/her notes are being stored. Instead, the user just treats each note (or collection of notes) as an object which can be manipulated in various ways.

Relish follows the standard Presentation Manager object-action model and the 1989 Common User Access advanced interface design guidelines. The user selects a note (usually from the main window) and then uses the menu options to perform one or more actions upon it. In addition, the user may change views to see different sets of notes (or to see the same notes in a different way). For instance, the Daily, Weekly, and Monthly schedules are all views. So are the To Do and Overdue lists.

Figure 5 shows two Relish windows on the desktop. One contains a Daily schedule with a graphical overview of committed time as well as a list summarizing each different note. The other contains the To Do list, organized in priority order (and by date and time due within each priority).

Relish also provides the user with a method for establishing other views using a query-by-example strategy (via the Lookup menu). The user can overlay these collection mechanisms by defining a set of categories (called *groups*) and assigning individual notes to these various categories. Typically, different groups are used for different activities or different tasks. For example, a user working on three different projects could use three groups to capture the events related to each project. When the user selects a particular group (via the Choose Group menu option), only notes in that group appear in his/her daily schedule, etc; when no group is selected, all notes appear. This provides an easy way for the user to view everything related to a particular activity, yet still be able to have a complete schedule for time planning and resolution of conflicts. It also impacts the overall design of Relish since the user can view different groups in different windows — all at the same time.

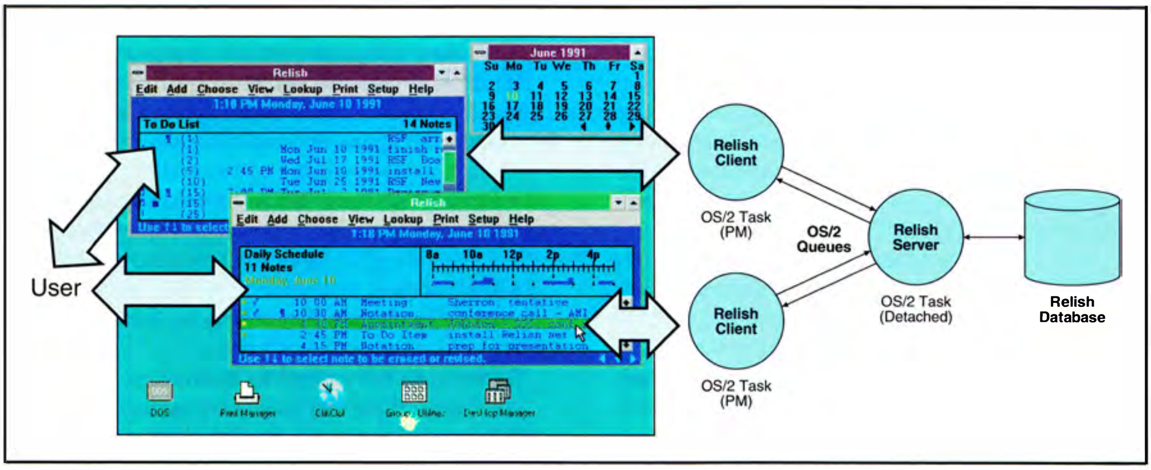
Relish uses a client-server architecture even in its single-user version. The client is responsible for all user-interface issues while the server is responsible for all data management issues, including control of the Relish database. Figure 6 shows the client-server partitioning that underlies the Daily schedule and To Do list shown earlier. One copy of the client program is managing the user's interaction with the window showing the Daily schedule; another copy is managing interaction with the window showing the To Do list. Additional clients would be added to the picture if other schedules or specific groups of notes were being viewed at the same time.

CLIENT-SERVER ARCHITECTURE IS A GOOD DESIGN MODEL

In designing a new application, early commitment to client-server architecture can produce significant benefits in both the short and long term. Some of the major ones, and how they apply to Relish, are discussed next.

Encourages Design Partitioning

One major benefit of designing an application in terms of a client-server architecture is that it encourages good design partitioning. The two (or more) programs which make up the application don't share code or data in any direct way. Each is entirely isolated.



```
   ¶ (1)                              RSF  arr
     (1)          Mon Jun 10 1991 finish r
     (2)          Wed Jul 17 1991 RSF  Boo
     (5)  2 45 PM Mon Jun 10 1991 install
     (10)         Tue Jun 25 1991 RSF  New
   ¶ (15)
     (15)
     (25)
Use ↑↓ to select
```

```
✓     10 00 AM  Meeting:      Sherron: tentative
✓   ¶ 10 30 AM  Notation:     conference call - AMI
       1 30 PM  Appointment:
       2 45 PM  To Do Item    install Relish net
       4 15 PM  Notation      prep for presentation
Use ↑↓ to select note to be erased or revised.
```

Figure 6. Relish Client and Server Tasks Supporting Two Relish Windows

The Relish database is managed as a single, shared resource by the server program. On any particular system, only a single copy of the server runs at a time. Whenever the user adds or revises a note, the server immediately becomes aware of the change. This, in turn, allows other clients to become immediately aware. Thus, if the user adds a To Do list item though the Daily schedule window, it immediately appears in the To Do list window as well.

The client and server approach provides the Relish user with other benefits. As soon as a note is entered, it is automatically saved in the database; the user never needs to explicitly save his/her schedule. The server also monitors all notes to be sure reminders are displayed at the appropriate times; the client need never get involved. This means that reminders are displayed even if Relish isn't *open* (as a window or icon) on the Presentation Manager desktop.

Thinking about application design in this way forms the basis for a *divide and conquer* design strategy. While designing either side of a client-server application is seldom an easy task, it is usually far easier than designing the entire application *en toto*. By breaking up the design task, each piece becomes easier to manage. Each can be designed, implemented, and (to some degree) tested independent of the others, once the scope of the client and server are each decided upon. This applies just as much to single-user as it does to multi-user applications.

During the early part of the design process, this scoping can be accomplished by taking the various functions provided by the application and assigning them to one program or the other. Part of the partitioning is usually quite easy due to the basic division of labor found in many client-server systems, including Relish:

- User interface functions are handled by the client.

- Data management functions are handled by the server.

In Relish all user interaction functions involved in entering and editing *notes* are done by the client. All functions related to storing and organizing the notes in the database are done by the server.

Don't just settle for the first solution that comes to mind.

In most cases, it is necessary to divide some functions into components, and then assign each to a different program. While Relish was being designed, the following basic function was identified:

- Give the user a reminder whenever the alarm time in a note is reached.

The client-server architecture necessitated division of the function into two others:

- Determine when the alarm time of a note is reached.

- Display a reminder to the user.

In this case, the former was assigned to the server (since it is a data management function) and the latter to the client (since it is a user interface function).

There are often several different ways to divide a function into components. Some are better than others and many factors can affect the decision. But one particular characteristic of client-server architecture often plays the key role:

- Client-server communication costs...

Whenever the client and server need to communicate, it raises the cost of the system. This is true for just about every kind of cost involved in building an application. It costs system resources — switching between the client and server tasks takes time; using the interprocess communication facilities takes time and (usually) memory. It costs design resources — both people and time are required to design the protocol that the client and server use to communicate; more communication takes more people time and more calendar time.

This is not to say that a good client-server architecture has a minimum number of the simplest possible client-server interactions. On the contrary, such a strategy may also minimize the beneficial aspects of the architecture. What it says, however, is that time and effort should be spent on evaluating alternatives when drawing the dividing line between client and server. More than likely, it will turn out to be time well spent. Don't just settle on the first *solution* that comes to mind.

The same scoping process is true for data (and other resources) used by the application. Again, general rules make some of the partitioning easy:

- Persistent, long-term, or shared data is usually the responsibility of the server.

- Interactive, short-term, or private data is usually the responsibility of the client.

Some data may need to be broken into component pieces and assigned to the two different programs. In Relish, for example, the information used to display the list of summarized notes is stored in the client. The one-line summaries are of no particular value to the server; this presentation (interaction) data is used over the relatively short time that the user is displaying a particular view. On the other hand, the note from which a one-line summary is derived is the responsibility of the server. This is because that note is not only of long-term interest, but also always shareable by other client windows.

Encourages Early Interface Design

Once the basic scoping of function and data has been accomplished, effort must be spent to design a set of service requests and responses that the programs can exchange to get the job done. This is sometimes referred to as the client-server protocol. In Relish, it includes services to:

- Add a note to the database (AddNote).

- Get an existing note from the database (GetNote).

- Print a particular schedule (PrintSchedule).
- Register a particular view with the server so that updates will be provided when the contents of the view are changed (RegisterMonitor).

In total, there are about two dozen services used by the Relish client and five major categories of data exchanged between the client and server.

Some people think that designing the client-server protocol is extra work that wouldn't be required if the application was designed as a single program. But that's not really true. They might not take the same form, but the interfaces these services and responses represent would appear *somewhere* inside that single program. Many probably wouldn't be considered in any detail, however, until far later in the design and implementation cycle. And that's another benefit of applying client-server architecture to single-user systems — these key interface issues are raised earlier in the design process when any problems they pose are easier to resolve.

Developing the client-server protocol isn't a simple task. And, during the process, it's best not to consider the previously decided scoping as cast in concrete. It may make more sense to reassign responsibility for a given function or a given data element than to provide the necessary services for maintaining it.

For example, one set of information maintained by Relish is that related to the initial sizes and positions of its various windows. (Relish saves this information whenever told to do so by the OS/2 Desktop Manager or whenever its own Choose Workspace menu option is used.) This is clearly persistent, long-term information that might even be shared by several clients. By the general design rules, it should be the responsibility of the server. Services such as StoreWindowState and GetWindowState could be defined in the client-server protocol to support the necessary exchange of information. However, that's not how Relish does it.

Rather than exchanging the information between the client and the server, the function for managing this long-term data was assigned to the client. There were several reasons for the decision, not the least of which was that this information is solely involved in the definition of the particular user interface, i.e., Presentation Manager, and even the current display being used. As the experienced OS/2 developer might expect, this information is stored instead in the OS/2 profile and accessed directly by the client when needed.

Spending time on the scoping and protocol early in the design process can lead to innovative solutions. A traditional design, for example, would require the user to explicitly perform a save operation before a note could gain long-term persistence (and visibility) on a schedule. The Relish design, on the other hand, resulted from careful examination of what needs to happen to a note. Thus, as soon as a user completes a new note (by clicking on OK in the Add Note dialog), responsibility for the note is transferred from the client to the server. It is at that point that the character of the data changes– from a collection of values being entered and edited by the user into a single entity of long-term value to the user. The server then saves this note and can inform any clients that need to know about it (such as a client presenting the Relish To Do list in one window when the user has just entered a To Do note onto a Daily schedule in another).

It makes it easier for the program to evolve over time.

Client and Server Evolution

Another major benefit to the use of client-server architecture is that it makes it easier for the program to evolve over time. Typically, the factors driving the evolution of the client and server components are different. Client evolution is often driven by changes in user interface needs, to support different user interactions or different user interface technologies. Server evolution is often driven by changes in data management technology or the need to change the form in which information is stored. By keeping the user interface and data management issues separate, incremental change is often easier to accomplish than if the entire application were written as a single program.

For some applications it may also make sense to provide several clients with different interface characteristics. A product providing search capabilities on some complex database might, for example, include one interface for expert users who prefer to interact using commands in some query or macro language as well as another interface for casual users who prefer to ask questions or *fill in the blanks* concerning what they are looking for.

In other situations, multiple clients may provide interfaces related to technology or tool differences. For instance, the server used in Relish actually supports two different clients (with basically the same functionality). The second (it came later) is the one described in this article. The first is a character-based implementation designed primarily for use with OS/2 1.0 (and is not part of the current retail product). This strategy actually allowed much of the Relish server to be developed and tested before any Presentation Manager tools were available. Further, it allows continued experimentation with additional interfaces without the need to redesign (or reimplement) the entire application.

Similar logic applies in the case where the server itself changes. The same client might be capable of interacting with different versions of the server. For example, one server version might use a proprietary database technology to manage shared data and another might use a standardized one such as the OS/2 Extended Edition Database Manager; in either case the client can remain unchanged.

Further, the use of client-server architecture for a single-user application provides a sound foundation when it comes time to extend that application for use in a multi-user environment (such as with IBM Lan Server or Microsoft Lan Manager).

Consider the following dilemma. In the multi-user version of Relish, assume there are two users named Scott and Carla. While Scott is adding a note to his schedule on one system, Carla might be viewing Scott's schedule on another system. Once Scott finishes adding the note, Carla's view of the schedule needs to be updated too.

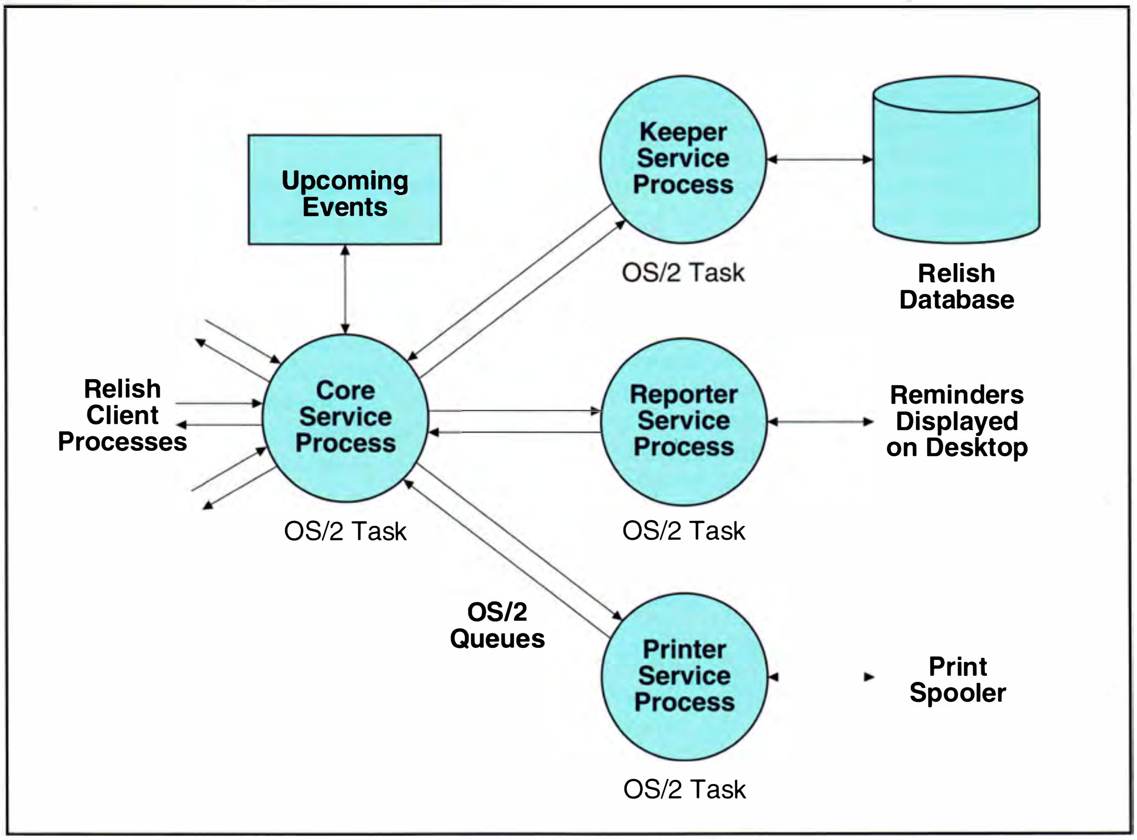


Figure 7. Tasks Which Form the Relish Server

If Relish had originally been designed without a client-server architecture, adding this feature to the multi-user version would have been a major design problem. Coordinating two programs on two systems to know that an action taken by one needs to be reflected in the other can mean major surgery for a traditional, single-user application.

But the client-server design partitioning in the single-user version of Relish made the change quite easy. Since a note becomes a persistent, long-term object passed to the server as soon as it is created, it is easy to provide other users access to it just after creation. The only difference between updating another Relish window on the same system and one on a remote system is the underlying *plumbing* that connects the client(s) to the server.

A Broader View of Client-Server Relationships

All the client-server relationships considered thus far have been *simple*; one program is obviously the client and the other is obviously the server. While many client-server applications fit this mold today, it is unlikely that things will stay simple for very long. A server may make use of other servers to perform part of its assigned functions.

Consider the internal design of the Relish server. The same functional and data scoping process applied to the division of labor between the Relish client and server processes was applied to the internal structure of the Relish server as well. The result, shown in Figure 7, consists of four separate tasks (or processes):

- The *Core Service Process* maintains a memory-resident list of all notes which will require action (such as displaying a reminder) in the near future (typically, the next few hours). It also has global responsibility for managing interprocess-communication among the client and server tasks; it acts as the point of contact for each client with the server as a whole.

- The *Keeper Service Process* maintains the disk-based database. It is the only process allowed to access the files that actually form the database and, as such, coordinates access to that resource by serializing requests.

- The *Printer Service Process* handles print (and export) requests (typically initiated by an ordinary client). It obtains the necessary data from the Keeper Service Process (via the Core Service Process), formats the information based on parameters in the request, and routes the output to a printer, file, or the system clipboard.

- The *Reporter Service Process* displays reminders and lets the user take action upon them. Thus, when the Core Service Process determines it is time to display a reminder, it sends a service request to the Reporter Service Process to carry out the action.

Each of these processes is a *server* with responsibility for providing particular services to *clients*. But in this case, the Core Service Process is the client while the other parts are the servers.

The service processes interact by sending service requests and by receiving responses just as in any other client-server relationship. The interactions follow the same protocol as the requests made by ordinary clients to the server as a whole. This allows, among other things, routing of incoming requests to the service process with ultimate responsibility for fulfilling the request; a GetNote request is routed to the Keeper Service Process while a PrintSchedule request is routed to the Printer Service Process. (As one might expect, however, there are some service requests that are used solely among these cooperating service processes and never by ordinary clients.)

A server may make use of other servers to perform part of its assigned functions.

In addition, the Reporter Service Process raises another interesting issue. Since it interacts with the user but is considered to be part of the server, it apparently violates the client-server division of labor. From the standpoint of the Core Service Process, it's a server since it carries out actions to display reminders and such. But from the point of view of the overall application, it's a client since it interacts with the user and, in fact, lets the user make modifications to a note in the same way as an ordinary Relish client. So which is it?

It's best to think about the Reporter Service Process as *both* a client and a server. There is no hard and fast rule in client-server architecture that requires a particular program to be one or the other. For any given use of any given service, one side is making the request and the other side is providing the response; thus one side is the client and one side is the server. But the same program might make requests for one service and provide responses to another. And the Relish Reporter is a limited example of just such a situation.

The same program might make requests for one service and provide responses to another.

Support for Broad Application Connectivity

Taking the broad view that the same program may act as both a client and a server leads to perhaps the most important benefit client-server architecture has for single-user applications. In the future, cohesive flexibility will allow any given application to both provide services to other applications and make use of services provided by others. Thus, an application will specialize in what it does best; it won't need to provide functionality that is best handled by other applications.

As computing power on the desktop gets more powerful, users are going to demand easier ways of accomplishing their daily activities. Future *solutions* to business needs will be more *user* centered and less *program* or *tool* centered. This means that the user preparing a complex report won't need to think about using:

- Program A to access some remote database,
- Program B to analyze the data and reduce it to meaningful numbers,
- Program C to build a spreadsheet to forecast results,
- Program D to format the report into a presentable form,
- Program E to generate the corresponding presentation,
- Program F to schedule the presentation,
- Program G to electronically distribute the report,
- Program H to...

Instead, the user will manipulate the components of the task at hand (the analysis, the spreadsheet, the presentation graphics, the meeting room schedule, etc.), in whatever way suits his/her method of accomplishing the entire task. The programs will need to interact to provide pieces of the puzzle. This will be true whether they are provided by one vendor, by multiple vendors, or written for in-house use by corporate developers.

Most likely, this collection of programs will interact in client-server terms. Each may (or may not) be a single-user application (in the traditional sense), but each might call on other programs for service and each might be called upon to provide service to others.

Thus, client-server architecture will become useful not only within particular applications, but also among those applications as well. To play the game in the new era of user-centered computing will require that a new application both provide services to and receive services from other applications.

IMPLICATIONS

Client-server computing doesn't always mean systems or programs connected across a LAN. It quite simply means programs cooperating with each other to accomplish a larger goal. One program asks the other to provide some information or to take some action — and it doesn't matter whether they are on the same or different systems.

Client-server architecture can provide substantial benefits specifically for single-user applications:

- It can help structure the design process by dividing it into smaller, more manageable tasks.
- It can lead to designs which are easier to implement since less internal program coordination may be involved.
- It naturally supports independent evolution of the client and server as the technologies driving their individual designs change.
- It can ease the transition between a single-user and multi-user design.

And, perhaps most important of all, client-server architecture may also open up new frontiers as programs begin to supply each other with the services they can do best.

REFERENCES

IBM. *Systems Application Architecture, Common User Access: Advanced Interface Design Guide.* (SC26-4582-0)

IBM. *Systems Application Architecture, Writing Applications: A Design Guide.* (SC26-4362-1)

Jackson, Bryan. "A Multi-User Server Using Named Pipes." *IBM Personal Systems Developer*, Winter 1990. 38-45pp.

Lafore, Robert, and Norton, Peter. *Peter Norton's Inside OS/2.* Brady Books, 1988

Letwin, Gordon. *Inside OS/2.* Microsoft Press, 1988.

Orfali, Robert, and Harkey, Dan. *Client-Server Programming with OS/2 Extended Edition.* Van Nostrand Reinhold, 1991.

Ryan, Ralph. *The Microsoft LAN Manager: A Programmer's Guide.* Microsoft Press, 1990.

Randell S. Flint, *Sundial Systems Corporation, 909 Electric Avenue, Suite 204, Seal Beach, CA 90740. (213) 596-5121, (310) 596-5121 after November 2, 1991. Dr. Flint is President and co-founder of Sundial Systems Corporation. He has been a consultant to many software development organizations in the areas of database technology, human-computer interface design, and software engineering. Dr. Flint was formerly a professor of Computer Science at California State University, Fullerton. He holds a Ph.D. in Information and Computer Science from the University of California, Irvine.*

SUNDIAL SYSTEMS

Systems Application Architecture

Cooperative Processing Across SAA Environments

Rajah Chacko

Leo YuHsiang Liu

Steve Waleski

by Rajah Chacko, Leo YuHsiang Liu, and Steve Wasleski

Cooperative Processing allows sharing and utilization of resources between machines. These machines can run under different operating systems including OS/2, VM/SP 6™, MVS™, or OS/400™. A cooperative application executes with portions distributed across two or more connected systems, one of which is a Programmable Workstation (PWS). Cooperative processing helps each machine perform its most efficient tasks.

IBM's Common Programming Interface-Communications (CPI-C) is a strategic interface within the Systems Application Architecture™ (SAA). Many programmers are turning to CPI-C™ since it is the ideal vehicle for implementing cooperative processing in SAA environments.

This article explains the rationale for splitting an application between the PWS and the host. It then shows how to use CPI-C to do a simple transaction and why CPI-C is preferred for cooperative processing in SAA environments. A second article compares CPI-C with APPC and describes a sample CPI-C program.

WHY COOPERATIVE PROCESSING?

Imagine you manage the Information Systems department of a growing insurance company. Your agents have PS/2s on their desks, but use them exclusively as terminals for accessing the mainframe. At 2:00 each afternoon, when the insurance quotations come streaming in, the mainframe slows to a crawl. The agents are over-utilizing their mainframe power but under-utilizing their PS/2 power. As new agents join the department, they struggle to learn the commands on the mainframe, wish they could use a graphics interface, and want to use pointing devices such as a mouse.

In this hypothetical situation, cooperative processing could meet your needs. Cooperative processing lets machines communicate at a higher level than terminal emulation: it conveys complex data conversations. Your systems programmers could write a small screen handler on the Programmable Workstation (PWS) with an extensive application on the host. Or they could write a cooperative application for the PWS and the host in which the PWS performs some calculations and the host portion performs database retrievals.

Either way, cooperative processing would combine the PS/2s with other hosts (machines running under OS/2, VM/SP 6, MVS, or OS/400). It would connect related tasks and resources among independent machines. And it would allow each machine to do what it does best, for example displaying a graphical user interface (on the PS/2), or accessing a large database (on the host). Cooperative processing could also provide a means to share expensive equipment, such as high-speed plotters.

It's easy to distinguish centralized processing from cooperative processing. The former concentrates all the work on the host, from moving cursors to retrieving data. The latter distributes the tasks between the PWS and the host.

HOW TO SPLIT AN APPLICATION

In the previous scenario, the PWS provides the better user interface, particularly with

Common User Access (CUA) dialogs, graphics, mouse, and fast response times. It can also perform many calculations, such as policy value projections. On the other hand, the host provides a powerful engine for accessing large databases, performing complex computations, and connecting many users. This means customer data and claims information can be accessed by many agents from one host. This host could also maintain bulletin boards for agents at different sites or in the field.

Figure 1 illustrates several ways to split an application between a PWS and a host. In the first case, the host performs all the tasks, and the PWS emulates a terminal. In the second case, the PWS performs the dialogs with graphics. The agent could use the mouse to interact with CUA dialogs. In the third case, the PWS performs simple data processing in addition to graphics. This case gives the agent fast response time even when the host is at its peak usage.

LOGICAL UNITS (LUs)

A Logical Unit (LU) is a piece of software that an application invokes to perform communications. It lies between the hardware and the application. LUs keep the programmer from having to set up buffers for data transmission, load hardware registers, change line status, maintain pointers and the like.

There are many different LU types for communications between applications, I/O devices, and subsystems. In OS/2, the two most commonly used LUs for cooperative processing are LU 2 and LU 6.2. LU 2 is used often for terminal emulation, while LU 6.2 is used primarily for application-to-application communications.

Although either LU 2 or LU 6.2 can be used for cooperative processing between a PWS and a host, their methodologies differ greatly. LU 6.2 acts like a telephone call: two partners establish a conversation and transfer information as needed. LU 2 works like a messaging system: the PWS leaves messages in the keyboard buffer of the host, and the host leaves messages in a virtual display buffer of the PWS. This is also how LU 2 performs terminal emulation: the user's keystrokes are placed in the keyboard buffer while a screen shows the contents of the display buffer. Developers have exploited LU 2 to allow operations like file transfer, which moves data through the keyboard buffer as if someone were typing very quickly.

Of the two Logical Unit types, LU 6.2 can adapt more quickly than LU 2. For instance, it can recover more quickly from errors such as a noisy line or a host going down. If the host went down while transferring a file using LU 2, the crash could hang the PWS for some time.

APPC AND CPI-C

Programming interfaces for LU 2 and LU 6.2 make them easier to use. One

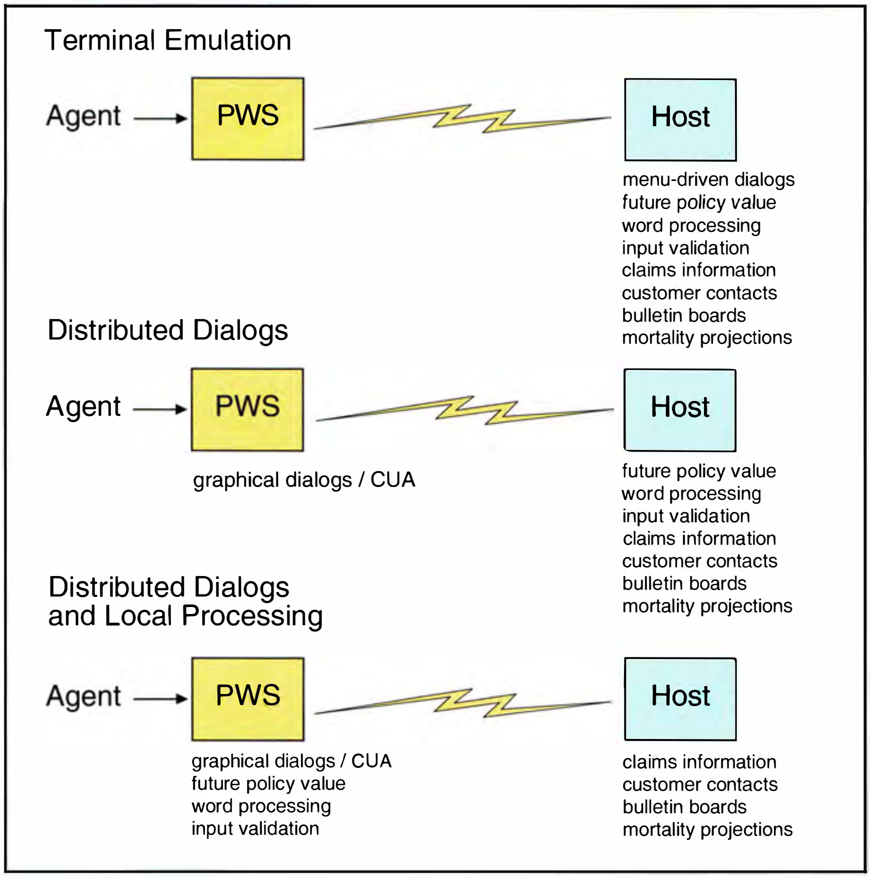


Figure 1. Splitting of Tasks Between a PWS and a Host

implementation of LU 2 is Emulator High Level Language Application Programming Interface (EHLLAPI, discussed below). Two implementations of LU 6.2 are Advanced Program-to-Program Communications (APPC) and CPI-C. Figure 2 shows the advantages for using APPC or CPI-C.

What makes CPI-C (or its basis, APPC) special?:

1. It can start programs on remote machines. This feature is an integral part of the protocol, implemented by the *attach manager* in OS/2 or the *resource manager* in other environments.

2. It handles cheap wires well. When bandwidth is inexpensive, such as with LANs, it allows many concurrent sessions, and rapid sending and receiving of data blocks.

3. It handles expensive wires well. When using long-distance telephone lines, for example, it collects all data into internal buffers, makes the call, sends all the data, and then hangs up as soon as possible.

4. It handles flooding data well. Data pacing can keep fast machines from drowning slow machines.

5. It handles multiplexing well. Many messages can be routed through one machine to many locations, all sequenced correctly.

6. It offers integral security. Password access, restricted user sets and data encryption are all available as part of the protocol.

7. It is available on almost any platform, including Apple™, Sun™, Novell and DEC™.

Courtesy of John Q. Walker II. Mr. Walker manages one of the development teams responsible for the implementation of APPC, APPN, and CPI-C for OS/2: IBM SAA Networking Services/2.

Figure 2. Reasons for Using APPC or CPI-C

LU 6.2 acts like a telephone call: two partners establish a conversation and transfer information as needed.

CPI-C is IBM's strategic Common Programming Interface (CPI) in SAA environments. It provides a rich, high-level set of constructs to help two computers communicate. CPI-C allows a computer to:

1. Establish a session with another computer.

2. Exchange data with the other computer.

3. Recover from errors in communication.

4. End the session.

These constructs help in resource management, whether the remote computer is managing a physical resource (such as a printer) or an abstract resource (such as a database). Figure 3 shows a simple example of how two machines can communicate with each other using CPI-C.

The constructs in Figure 3 are very high-level, but they make a great deal of sense: one machine establishes a session and conversation; the other accepts the conversation. Data flows between the two machines, and they take turns sending and receiving. When the processing is completed, the machine in control of the conversation ends the session.

A human analogy is the process of making a phone call: one person calls another (establishes a session), identifies himself and his business (establishes conversation); they take turns talking to each other (send and receive), say good-bye and hang up (end the session).

If CPI-C is not available in an environment, programmers can take two routes to use APPC with other systems that use CPI-C. First, they can write APPC on the PWS and carefully match CPI-C verbs. Second, they can emulate CPI-C using APPC. The latter is fairly easy to implement and greatly simplifies the communications programming process.

THE CPI-C BENCHMARK

The following are other communication services that are available in the OS/2 environment. Each section introduces the product, discusses its underlying protocols, and gives a brief comparison with CPI-C. Figure 4 summarizes the differences among these communications products.

APPC: Advanced Program-to-Program Communication lets the programmer completely control most aspects of the conversation, such as conversation security. It is available under VM, MVS, OS/400, OS/2 and DOS operating systems. The shortcoming of APPC is its lack of portability: it works differently on different systems. For instance, the programmer passes a pointer to parameter block under OS/2, but calls the APPC routine under VM/SP with a different order of parameters.

NetBIOS: NetBIOS runs on a token-ring network and can be used only from OS/2 or DOS, so it excludes hosts. It uses neither LU 2 nor LU 6.2. NetBIOS also lacks certain features that aid the programmer in processing complex transactions, such as conversations and transmission confirmations. Creating applications in NetBIOS with these features would be much more complex than the equivalent CPI-C program.

SRPI: The Server-Requester Programming Interface (SRPI) offers powerful constructs for sharing information between a PWS and a host. Its features allow the programmer to use the host as a PWS virtual disk or virtual printer and to copy files between the host and PWS. In addition, SRPI offers host database services to access DB2 or SQL/DS databases from the PWS. Like most LU 2 products, if a host fails during a file copy, the failure could hang the PWS for some time. The PWS would have to wait until a timer expired.

SRPI shares the NetBIOS disadvantages of not being compatible with SAA and not being available on all hosts. It also restricts the PWS to be a client and the host to be a server, while CPI-C will allow any machine to be client or server.

EHLLAPI: The Emulator High Level Language Application Programming Interface (EHLLAPI) operates like SRPI, through LU 2. The strength of EHLLAPI is its ease-of-use, in that it allows the programmer's application to both simulate keystrokes to an existing 3270 session, and to read the display buffer from the host responses. EHLLAPI shares the LU 2 problems in case of a host failure and works best with a predetermined dialog. It is better suited to automating repetitive tasks on the host.

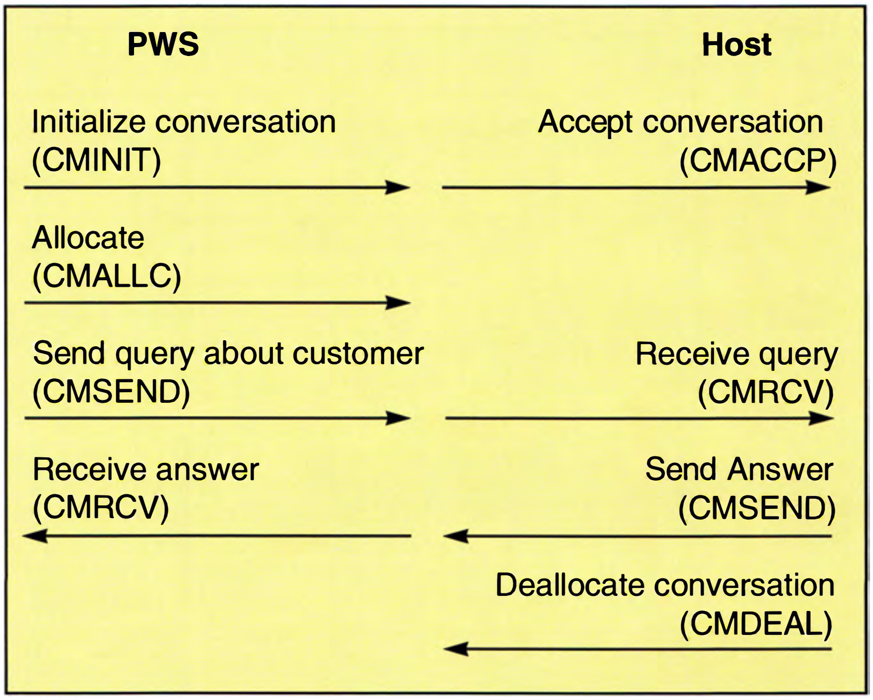

PWS
Host
Initialize conversation (CMINIT)
Accept conversation (CMACCP)
Allocate (CMALLC)
Send query about customer (CMSEND)
Receive query (CMRCV)
Receive answer (CMRCV)
Send Answer (CMSEND)
Deallocate conversation (CMDEAL)

Figure 3. Sample CPI-C Data Exchange Between the PWS and Host

Product	SAA?	Programming Complexity	Underlying Protocol
CPI-C	Yes	Low	LU 6.2
APPC	No	High	LU 6.2
NetBIOS	No	High	NetBIOS
SRPI	No	Low	LU 2
EHLLAPI	No	Low	LU 2

Figure 4. Summary of Communications Products

ACKNOWLEDGMENTS

The authors wish to thank Delores Hamilton for her editing, Anna Ottobelli for her help in connecting OS/2 to VM, Randy Miller for his comments about NetBIOS, and John Q. Walker II for his insightful technical comments.

REFERENCES

IBM. *Local Area Network Technical Reference.* (SC30-3383).

IBM. *OS/2 Extended Edition Version 1.3 APPC Programming Reference.* (S01F-0295).

IBM. *OS/2 Extended Edition Version 1.2 Cookbook: Communications Manager Design and Implementation.* (GG24-3552).

IBM. *OS/2 Extended Edition Version 1.2 Cookbook: Communications Manager SNA Environment.* (GG24-3553).

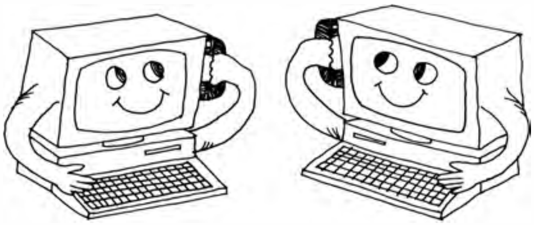
IBM. *Program-to-Program Communications in SAA Environments.* (GG24-3482).

IBM. *SAA CPI Communications Reference.* (SC26-4399).

IBM. *Virtual Machine/System Product Connectivity Programming Guide and Reference.* (SC24-5377).

Libutti, L. Robert. *Systems Application Architecture, The IBM SAA Strategy,* IBM Corporation (GC26-4784).

Scherr, A. L. "SAA Distributed Processing." *IBM Systems Journal,* (Vol. 27, No. 3, 1988).



Rajah Y. Chacko, *IBM Programming System Lab, 11000 Regency Parkway, Cary, NC 27512-9968. Mr. Chacko edits the Cary Technical Vitality Newsletter. He joined the Cross System Product Advanced Applications Generator Development Department, Cary, North Carolina, in 1989. He holds an BS in Mathematics/ Computer Science from Carnegie Mellon University, and is working toward an MS in Computer Science at Duke University.*

Leo Y. Liu, *IBM Programming System Lab, 11000 Regency Parkway, Cary, NC 27512-9968. Dr. Liu joined the Cross System Product Advanced Design and Strategy Department, Cary, North Carolina, in 1989. He is responsible for the support of cooperative applications for the IBM Application Generator, Cross System Product. He holds a Ph.D. in computer science from Pennsylvania State University, and both an MS in Management Science and a BS in Electronic Engineering from the National Chiao-Tung University, Taiwan.*

Steve Wasleski, *IBM Programming System Lab, 11000 Regency Parkway, Cary, NC 27512-9968. Mr. Wasleski joined IBM in 1987 and is currently in the Cross System Product (CSP) Advanced Design and Strategy Department. His responsibilities include the support of cooperative applications and Common User Access (CUA) for the IBM Application Generator, CSP. He holds a BS in Computer Science from the University of Missouri-Rolla.*

Systems Application Architecture

A Comparison of CPI-C and APPC on OS/2

by Steven F. Wasleski, Leo YuHsiang Liu, and Rajah Y. Chacko

The Common Programming Interface for Communications (CPI-C) has advantages over Advanced Program-to-Program Communications (APPC) for SAA cooperative processing, not only because CPI-C is the IBM strategic CPI for communications, but more importantly because of its simplicity, portability, and interoperability across SAA environments.

CPI-C for OS/2 became generally available from IBM on March 29, 1991 as part of SAA Networking Services/2 (NS/2), the OS/2 Advanced Peer-to-Peer Networking (APPN) product. CPI-C was not available on OS/2 when the authors began their work last year. Fortunately, the International Technical Support Center (ITSC) had shown a simple emulated CPI-C interface using APPC on both OS/2 and OS/400. The authors first give a brief comparison between CPI-C and APPC and then provide a CPI-C version of the APPC example that appeared in the Summer 1989 issue of the IBM Personal Systems Developer.

WHAT IS CPI-C?

CPI-C is IBM's SAA communications CPI. Although CPI-C is currently available in only some of the SAA environments, it will be a portable LU 6.2 interface for all SAA platforms. Interested readers are referred to the System Application Architecture Common Programming Interface Communications Reference, SC26-4399, for more details.

Some Advantages of CPI-C

Since CPI-C has been chosen as the strategic direction, it will receive the most attention in the future in terms of increased functionality, performance, portability, and interoperability.

CPI-C's parameterized call interface will be portable across all SAA platforms. APPC, on the other hand, has very poor portability. The main reason is that the control blocks, passed to the APPC entry point, have very different layouts for different systems.

CPI-C's parameterized call interface is much easier for most programmers to understand than APPC's control block interface. Most programmers we know have been using procedure calls since they wrote their first *Hello, world* program, while very few programmers have experience with control blocks. Furthermore, call interfaces are typically less prone to error and easier to debug.

A COMPARISON OF CPI-C AND APPC

The most noticeable difference between CPI-C and APPC is how they are invoked. CPI-C uses a standard procedure call interface with one entry point for each call. APPC has two entry points that take a control block pointer as a parameter. You must first fill the control block with the appropriate values.

Steve Wasleski

Leo YuHsiang Liu

Rajah Y. Chacko

CPI-C Call	Entry Point	Corresponding APPC Verbs
Accept Conversation	CMACCP	CONVERT,RECEIVE_ALLOCATE
Allocate	CMALLC	MC_ALLOCATE
Confirm	CMCFM	MC_CONFIRM
Confirmed	CMCFMD	MC_CONFIRMED
Deallocate	CMDEAL	MC_DEALLOCATE,TP_ENDED
Flush	CMFLUS	MC_FLUSH
Initialize Conversation	CMINIT	CONVERT, TP_STARTED
Prepare To Receive	CMPTR	MC_PREPARE_TO_RECEIVE
Receive	CMRCV	MC_RECEIVE_AND_WAIT, MC_RECEIVE_IMMEDIATE
Request To Send	CMRTS	MC_REQUEST_TO_SEND
Send Data	CMSEND	MC_SEND_DATA
Send Error	CMSERR	MC_SEND_ERROR
Test Request To Send Received	CMTRTS	MC_TEST_RTS

Table 1. Related APPC Verbs for a CPI-C Call

While the interfaces are radically different, the functions provided by CPI-C and APPC are extremely similar. The next two sections provide tables that demonstrate the correlation of CPI-C to APPC in two major areas, verbs and states. Other areas, such as return codes, parameters (APPC control block entries verses CPI-C conversation characteristics), and configuration requirements are also very similar, but are not covered here. Most conceptual APPC knowledge is transferable to CPI-C programming.

CPI-C Calls and APPC Verbs

Table 1 shows the CPI-C calls and the APPC verbs we used to emulate them. We wrote a more robust CPI-C emulation based on the ITSC version for our own use last year. Our emulation only supports mapped conversations, since that was all we needed. Therefore, the table uses the mapped (MC_) APPC verbs. The CONVERT, RECEIVE_ ALLOCATE, TP_STARTED, and TP_END verbs are common between mapped and basic conversations.

CPI-C includes four EXTRACT calls and thirteen SET calls that are not shown in the table. These CPI-C calls have no corresponding APPC verbs. They correspond to parameters stuffed inside the APPC control blocks. For example, instead of specifying a send type of flush on every MC_SEND_DATA verb, you call the Set_Send_Type verb, CMSST. CMSST is called once with the parameter CM_SEND_AND_FLUSH. All subsequent Send_Data verbs (until the next CMSST call) will use the flush send type. In our emulation, these values are stored in a table dynamically allocated during the CMINIT or CMACCP call. These calls initialize the tables to the appropriate values according to the CPI-C Reference. An EXTRACT call simply returns the current value in the table. A SET call changes the current value in the table.

Note that the synchronization-point changes in the latest version of the CPI-C reference are not addressed in this article. Although these changes are important additions to CPI-C, they are not needed to understand or use CPI-C and are left out to reduce the complexity of this article.

CPI-C and APPC States

Table 2 shows CPI-C and corresponding APPC states. You will notice that the CPI-C Initialize state has no corresponding APPC state. This is because APPC's state tables consider a conversation to still be in the Reset state after TP_STARTED. The subsequent MC_ALLOCATE call puts the conversation in the Send state. The CPI-C Initialize state is the second stage of the APPC Reset state for conversation characteristic modification.

The APPC Pending Post state is not used by CPI-C, since CPI-C does not have a receive_and_post characteristic. EXTRACT calls and the SET calls, CMSDT, CMSED, CMSF, CMSLD, CMSPTR, CMSRT, and CMSST, are valid in any state except Reset. The SET calls, CMSCT, CMSMN, CMSPLN, CMSRC, CMSSL, and CMSTPN, are valid only in the Initialize state. Therefore, this last group of characteristics may only be set by the conversation initiator, the program that issues CMINIT.

The CPI-C Reference provides a complete table of conversation states that describes how calls, parameters, and return codes interact to cause state changes. The number of combinations of these that can cause state changes are great, but after a little exposure, the pattern and logic behind the state changes are not too difficult to grasp. Also, if you are familiar with APPC, most of the state changes are the same.

A CPI-C SAMPLE PROGRAM

In the Summer 1989 edition of the IBM Personal Systems Developer, Sam Henderson and Jim Kovaric showed the APPC code (Figure 5 on pages 18-23) needed to transfer a block of data to a remote program. Figure 1, shows the same program written in CPI-C. To make comparisons easier, we will use the same sequence of communications steps and error detection. Some of the code and comments will be kept identical to help you synchronize the comparison. Note that our example does not accept or echo any command line parameters. Notice that all references to LUs, TP names, and ASCII to EBCDIC conversion of those parameters have been removed in the CPI-C program, because they are part of the symbolic destination name (SDN). This is configured outside the program and contains the needed LU and TP name information.

The filling of control blocks and the corresponding APPC calls are replaced by simple CPI-C calls.

CPI-C State	APPC State	CPI-C Calls that may be used in this CPI-C State
Reset	Reset	CMINIT, CMACCP
Initialize		CMALLC,CMDEAL (abend type only)
Send	Send	CMCFM, CMDEAL, CMFLUS, CMPTR, CMRCV (wait only), CMRTS, CMSEND, CMSERR, CMTRTS
Receive	Receive	CMDEAL (abend type only), CMRCV, CMRTS, CMSERR,CMTRTS
Send Pending	Send Pending	CMCFM, CMDEAL, CMFLUS, CMPTR, CMRCV (wait only), CMRTS, CMSEND, CMSERR, CMTRTS
Confirm	Confirm	CMCFMD, CMRTS, CMSERR
Confirm Send	Confirm Send	CMCFMD, CMRTS, CMSERR
Confirm Deallocate	Confirm Deallocate	CMCFMD, CMRTS, CMSERR

Table 2. CPI-C States and APPC States

```
/************************************************************************/
/*  This routine shows how to use CPI-C to start an application on      */
/*  a Partner LU and then transfer some data to that Partner.           */
/*  The following CPI-C verbs are used :                                */
/*                                                                      */
/*  CMINIT, CMSSL, CMALLC, CMCFM, CMSEND, CMDEAL                        */
/*                                                                      */
/*  The code example assumes the partner LU has been configured         */
/*  with a remote TP. This TP has been configured to accept             */
/*  mapped conversations and a sync level that accepts confirms.        */
/*                                                                      */
/*  The following string is sent to the partner LU:                     */
/*                                                                      */
/*  "PC6::C:\REPORTS\APRIL10.TXT"                                       */
/*                                                                      */
/*  This string instructs the remote application to get the file        */
/*                                                                      */
/*  "C:\REPORTS\APRIL10.TXT"                                            */
/*                                                                      */
/*  from another partner LU named PC6 and print that file.              */
/************************************************************************/
#define INCL_DOS
#include <os2.h>                        /* Found in OS/2 toolkit        */
#include <cpic.h>                       /* Interface to my emulation    */
#include <stdio.h>                      /* C library routines           */
#include <stdlib.h>                     /* C library routines           */
#include <string.h>                     /* C library routines           */


/************************************************************************/
/*  MAIN ROUTINE                                                        */
/************************************************************************/
extern void cdecl main()
{
SEL selector;
PUCHAR data_ptr;
UCHAR conv_id[8];
ULONG rc;
ULONG rts_rcvd;
PUCHAR print_report = "PC6::c:\\reports\\april10.txt";
/************************************************************************/
/*             Get a shared memory segment for data buffer.             */
/************************************************************************/
DosAllocSeg(512, &selector, 1);
data_ptr = (PUCHAR)MAKEP(selector, 0);

/************************************************************************/
/*          Initialize the conversation using the CMINIT verb.          */
/************************************************************************/
CMINIT(conv_id, "EXSDN ", &rc);

/************************************************************************/
/*                            Check for errors.                         */
/************************************************************************/
if (rc) {
printf(The CMINIT verb failed. RC: %1X", rc);
return;
}
```

Figure 1. Sample CPI-C Program (Continued)

```
/**********************************************************************/
/*                      Set up to use CMCFM verb.                     */
/**********************************************************************/
CMSSL(conv_id, CM_CONFIRM, &rc);

/**********************************************************************/
/*                          Check for errors.                         */
/**********************************************************************/
if (rc) {
printf(The CMSSL verb failed. RC: %lX", rc);
return;
}

/**********************************************************************/
/*        Allocate a conversation with a Remote application.          */
/**********************************************************************/
CMALLC(conv_id, &rc);

/**********************************************************************/
/*                        Check for errors.                           */
/**********************************************************************/
if (rc) {
printf(The CMALLC verb failed. RC: %lX", rc);
return;
}
/**********************************************************************/
/*                      Send the confirm verb.                        */
/**********************************************************************/
CMCFM(conv_id, &rts_rcvd, &rc);

/**********************************************************************/
/*                        Check for errors.                           */
/**********************************************************************/
if (rc) {
printf(The CMCFM verb failed. RC: %lX", rc);
return;
}

/**********************************************************************/
/*                      Issue the send data verb.                     */
/**********************************************************************/
memcpy(data_ptr, print_report, strlen(print_report));
CMSEND(conv_id, data_ptr, strlen(print_report), &rts_rcvd, &rc);

/**********************************************************************/
/*                        Check for errors.                           */
/**********************************************************************/
if (rc) {
printf(The CMSEND verb failed. RC: %lX", rc);
return;
}

/**********************************************************************/
/*                     Send the confirm verb.                         */
/**********************************************************************/
CMCFM(conv_id, &rts_rcvd, &rc);

/**********************************************************************/
/*                        Check for errors.                           */
/**********************************************************************/
```

Figure 1. Sample CPI-C Program (Continued)

```
if (rc) {
printf(The CMCFM verb failed. RC: %lX", rc); return;
}

/**********************************************************************/
/*            Deallocate the conversation with the partner.
*/
/**********************************************************************/
CMDEAL(conv_id, &rc);
```

Figure 1. Sample CPI-C Program

The Communications Manager and NS/2 configuration needed to make APPC or CPI-C work can be quite complex and will not be described in this article. The NS/2 configuration tools have simplified the process considerably.

A Comparison

Many differences can be found between the two programs. First, the reduction in both size and complexity of the code is remarkable. Second, for average programmers, the CPI-C procedure call interface is more intuitive than APPC control blocks. Third, the code in Figure 1 could easily be ported to another SAA platform, with little or no change. The APPC code would require extensive modification.

CONCLUDING REMARKS

Both CPI-C and APPC are implemented based on LU 6.2, so a CPI-C program can communicate with an APPC program without any extra work. Despite the differences in the syntax of verbs, states, and return codes, there is a natural functional correspondence between the two. It is quite easy for an APPC programmer to get familiar with CPI-C in a very short period. Furthermore, CPI-C is much easier to learn and use than APPC since CPI-C's callable interfaces hide APPC's control blocks, system dependencies, and parameter conversions from the programmers.

In this article, we have shown why CPI-C is preferred to APPC for SAA cooperative processing. The simplicity, portability, and interoperability of CPI-C make CPI-C a very attractive candidate for SAA cooperative processing.

ACKNOWLEDGMENTS

The authors would like to thank Anna Ottobelli of the Raleigh ITSC for her comments and CPI-C emulation code for OS/2.

REFERENCES

IBM, ITSC. *Program-to-Program Communications in SAA Environments.* (GG24-3482.)

IBM. *System Application Architecture Common Programming Interface Communications Reference.* (SC26-4399.)

Henderson, Sam and Kovaric, Jim. "Advanced Program-to-Program Communications (APPC)." *IBM Personal Systems Developer.* Summer 1989.

Steve Wasleski, *IBM Programming System Lab, 11000 Regency Parkway, Cary, NC 27512-9968. Mr. Wasleski joined IBM in 1987 and is currently in the Cross System Product (CSP) Advanced Design and Strategy Department. His responsibilities include the support of cooperative applications and Common User Access (CUA) for the IBM Application Generator, CSP. He holds a BS in Computer Science from the University of Missouri-Rolla.*

Leo Liu, *IBM Programming System Lab, 11000 Regency Parkway, Cary, NC 27512-9968. Dr. Liu joined the Cross System Product Advanced Design and Strategy Department, Cary, North Carolina, in 1989. He is responsible for the support of cooperative applications for the IBM Application Generator, Cross System Product. He holds a PhD in computer science from Pennsylvania State University,and an MS in Management Science and a BS in Electronic Engineering from the National Chiao-Tung University, Taiwan.*

Rajah Y. Chacko, *IBM Programming System Lab, 11000 Regency Parkway, Cary, NC 27512-9968. Mr. Chacko joined the Cross System Product Advanced Applications Generator Development Department, Cary, North Carolina, in 1989. He holds a BS in Mathematics/ Computer Science from Carnegie Mellon University, and is working toward an MS in Computer Science at Duke University.*

LAN Requester and Server

Programming to the OS/2 NetBIOS API

Dana Beatty

by Dana Beatty

The purpose of this article is to assist application programmers in writing to the NETwork Basic Input Output System (NetBIOS™) Application Programming Interface (API) provided with Operating System/2 Extended Edition. This is accomplished by first taking a high-level look at NetBIOS, then concentrating on steps necessary to use the API. It is intended that this paper supplement the technical reference publications available through IBM Branch offices listed at the end of this article.

PROGRAMMING TO THE OPERATING SYSTEM/2 NETBIOS API

Because NetBIOS is a high-level Application Programming Interface (API) it is not necessary to understand the intricacies of Local Area Networks (LANs) to write a program that runs over a LAN. The task of writing an application to this API can be expedited by focusing instead on specifics of NetBIOS and the nuances of OS/2 system implementation.

WHAT IS NETBIOS?

NetBIOS is a high-level, *name-oriented* API that uses the IEEE 802.2 interface of OS/2 Extended Edition to communicate across Local Area Networks. It provides an interface between an application program and the network that shields the application from the complexity of handling communication protocols.

NetBIOS provides two forms of data transfer: *datagrams* and *sessions*. With datagrams, data transfer is accomplished in a best-effort fashion. Data is sent directly to the IEEE link layer with no guarantee delivery or error recovery.[1] Sessions provide a more reliable data transfer. A session is a logical connection between any two network names.

An application interfaces to NetBIOS through Network Control Blocks (NCBs) that are passed to the API in a FAR CALL/RETURN fashion. An NCB is simply a data structure that contains information required for issuing a command across the interface. It is translated by NetBIOS into one or more IEEE 802.2 commands that are issued to the adapter.[2]

NetBIOS has four command types: general support commands, name support commands, datagram support commands, and session support commands.

GENERAL SUPPORT COMMANDS

General support commands are concerned with the basic network environment. Perhaps the most important general support command is the NCB.RESET, the mechanism an application uses for allocating and deallocating resources. A RESET must be the first command an application issues.

A process defines its environment and obtains resources, such as *sessions, names,* and use of *name-number-one,* with the RESET command. Processes cannot share resources,

[1] See *IBM Personal Systems Developer, Summer 1990,* "Programming to the OS/2 IEEE 802.2 API" for a more in depth discussion on the IEEE 802.2 Type 1 service.

[2] Refer to the *LAN Technical Reference (SC30-3383)* for a complete listing of IEEE 802.2 802.2 commands, or *IBM Personal Systems Developer,* Summer 1990, ibid, for more details on this API.

including the *name-number-one.* The resources that a process claims are allocated from the overall NetBIOS resource pool, defined at load-time (IPL), based on values specified in the Communications Manager configuration file.[3]

A RESET is also used by a process to relinquish resources. All sessions and names associated with the process issuing this *deallocate* RESET are deleted. OS/2 performs an implicit RESET if a process terminates before deallocating its resources. Other applications running concurrently are not affected by this RESET.[4]

NAME SUPPORT COMMANDS

As stated previously, NetBIOS is *name-oriented.* Applications communicate with each other across the network through *names* maintained locally in an internal *Name Table.* When a name is added to this table, NetBIOS returns a one-byte value known as the NAME NUMBER. This value is used in subsequent commands instead of the actual NetBIOS name.

A name may either be a *group name,* (for an entire department), or an *individual* name (for a member of a department). Group names facilitate the concurrent transfer of data to a number of workstations, while individual names permit data transfer to a single station.

DATAGRAM SUPPORT COMMANDS

These commands provide transfer of messages with no guarantee of delivery, acknowledgement of receipt, or error recovery. A session is not required to send datagrams. Datagrams may be broadcast, or sent to a specific station. In either case, a corresponding RECEIVE must be outstanding.

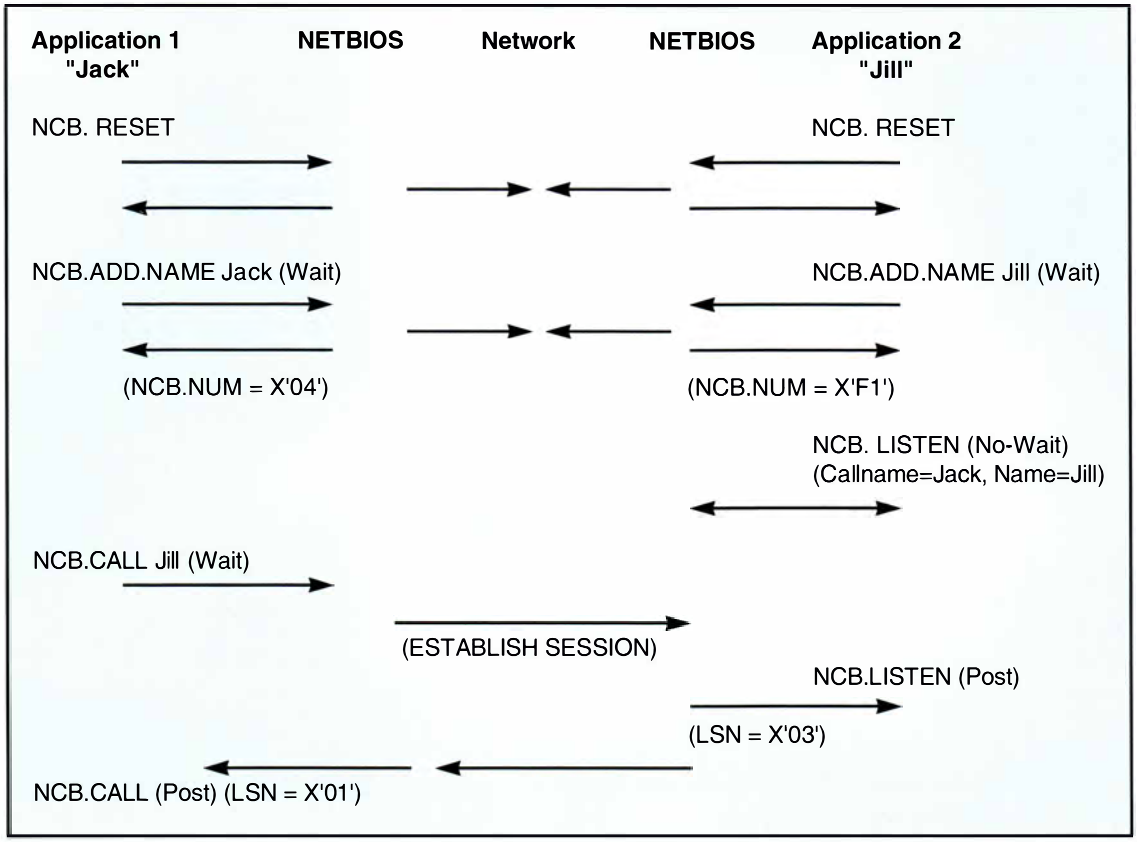


Figure 1. Establishing a NetBIOS Session

[3] The Configuration file is specified on the *CFG= parameter of the NetBIOS DEVICE= statement* in *CONFIG.SYS.* See also the *NetBIOS screen* in Figure 8.

[4] Note that any active sessions with the process that was *RESET* will terminate.

SESSION SUPPORT COMMANDS

The advantage of sessions over datagrams is that sessions provide two-way guaranteed delivery between two names. This guarantee is provided by the IEEE Logical Link Control layer, in addition to a NetBIOS level acknowledgement.

When a session is established, a local link station connects with a remote link station, causing a path, or *session*, to be created between two names. While there is a small amount of overhead associated with sessions, the benefits of guaranteed delivery often offset this.

To establish a session, one name on the network issues an NCB.LISTEN, while the other name issues an NCB.CALL as shown below in Figure 1. Once a session is created, NetBIOS returns a Local Session Number (LSN) to the application. The LSN is required for subsequent Send, Receive, and Hang-up commands.

SENDING AND RECEIVING DATA

A RECEIVE command must be outstanding for a name to accept data. There are three types of RECEIVE commands:

```
NCB.RECEIVE
NCB.RECEIVE.ANY(Specific)
NCB.RECEIVE.ANY(Any)[5]
```

An NCB.RECEIVE allows a name to receive data from the session partner denoted by the LSN. If more than one RECEIVE is outstanding, an NCB.RECEIVE has precedence over an NCB.RECEIVE .ANY(Specific) which, in turn, has precedence over an NCB.RECEIVE.ANY(Any).

If a RECEIVE is outstanding, a partner may send data by issuing one of the following SEND commands:

```
NCB.CHAIN.SEND
NCB.CHAIN.SEND.NO.ACK
NCB.SEND
NCB.SEND.NO.ACK
```

Figure 2 shows the data transferred over a session.

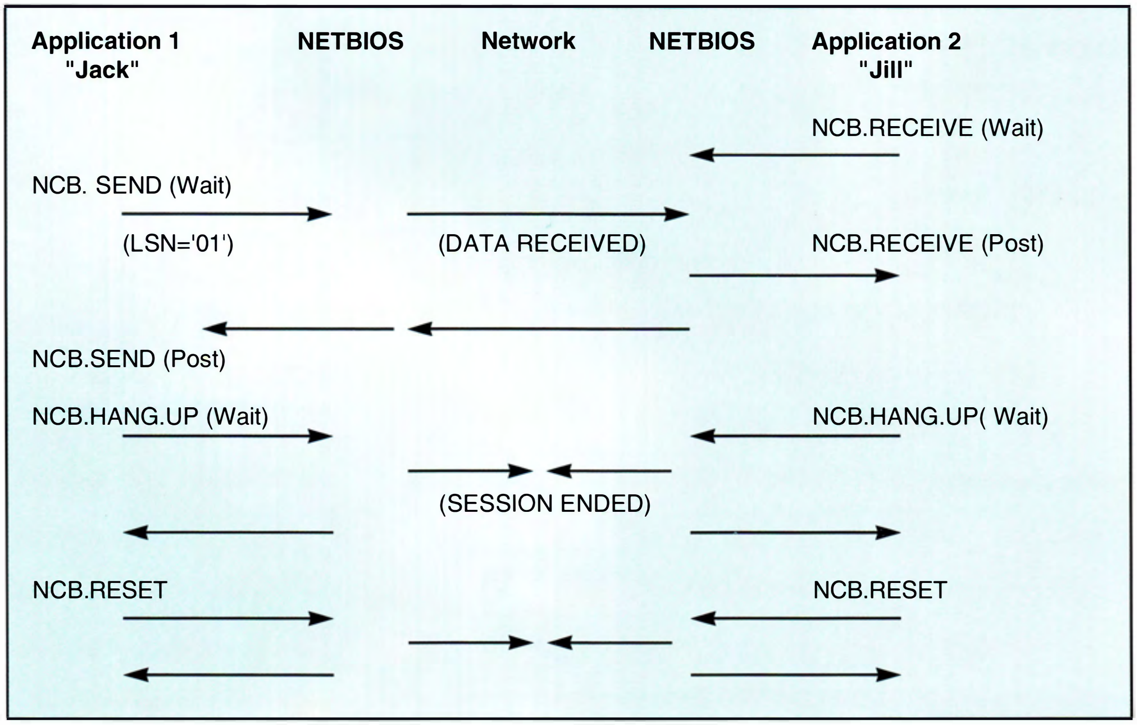


Figure 2. Sending/Receiving Data

[5] Caution should be used when issuing *NCB.RECEIVE.ANY(Any)*, as messages destined for other programs may satisfy this *RECEIVE*.

The CHAIN command permits data from two storage areas to be sent as a single message. There is not a corresponding "RECEIVE CHAIN" NCB, therefore if the receive buffer cannot contain the entire message, additional RECEIVEs must be issued.

Two new commands for the OS/2 system provide an application the option of sending messages without the additional overhead of NetBIOS level acknowledgements: NCB.CHAIN.SEND.NO.ACK and NCB.SEND.NO.ACK. The time required to complete a SEND is reduced by eliminating the NetBIOS acknowledgement.

WAIT VERSUS NO-WAIT OPTION

NetBIOS provides two forms of commands: WAIT and NO-WAIT.

WAIT commands cause the application program to wait, and return control to the application when the command completes. Therefore, the WAIT option is not valid for the Device Driver interface, except for NCB.RESET and NCB.CANCEL (which must complete before the application program can continue). Any other command issued to the Device Driver Interface as a WAIT command will complete with a return code of X'03': Invalid Command.

Three general commands, RESET, CANCEL, and UNLINK, are guaranteed to complete, but the remaining NetBIOS commands are not. Therefore, it's possible for an infinite loop condition to occur and the application programmer must take care to avoid such an occurrence.

WAIT commands are easily recognized by the setting of the command's high order bit to zero. NO-WAIT commands set the high order bit of the command to one.

NO-WAIT commands return control to the application program immediately. For the Dynamic Link Routine (DLR) interface, NetBIOS spawns a child thread that is blocked until the command completes. The NO-WAIT completes as it did in PC DOS NetBIOS, except that the application must issue a FAR RETURN rather than an INTERRUPT RETURN.

While the command is in process, a temporary return code (X'FF') is placed in the NCB. The application program must not alter the NCB during this time frame.

NO-WAIT permits several commands to be simultaneously outstanding, thus providing maximum throughput. However, NO-WAIT commands should be used with caution since OS/2 resources are required for spawning and managing threads. It is possible for a NO-WAIT command to fail because these resources have been exhausted.[6]

OS/2 EE NETBIOS API

As a result of the architecture, the NetBIOS API provided with OS/2 EE differs somewhat from its PC DOS predecessor. The OS/2 EE NetBIOS API permits the user to select from two programming interfaces: *Dynamic Link Routine* and *Device Driver.* Both interfaces support multi-thread, multi-process applications concurrently over a maximum of two adapters.[7]

Three media types are supported through the OS/2 EE NetBIOS AND IEEE 802.2 interface: IBM Token-Ring Network, IBM PC Network (baseband and broadband), and ETHERAND™ network. OS/2 EE NetBIOS API provides this rich set of functions, while maintaining compatibility with its PC DOS predecessor. This means that OS/2 NetBIOS permits PC DOS stations to co-exist on the network.

MIGRATION CONSIDERATIONS

Both interfaces employ a Network Command Control Block (NCB).An NCB is simply a data structure that contains information required for issuing a command across the interface to the adapter.

[6] *X'35' return code:* Required operating resources exhausted, retry later.

[7] An application program may simultaneously use both interfaces, however, it will appear as separate applications to the *API.*

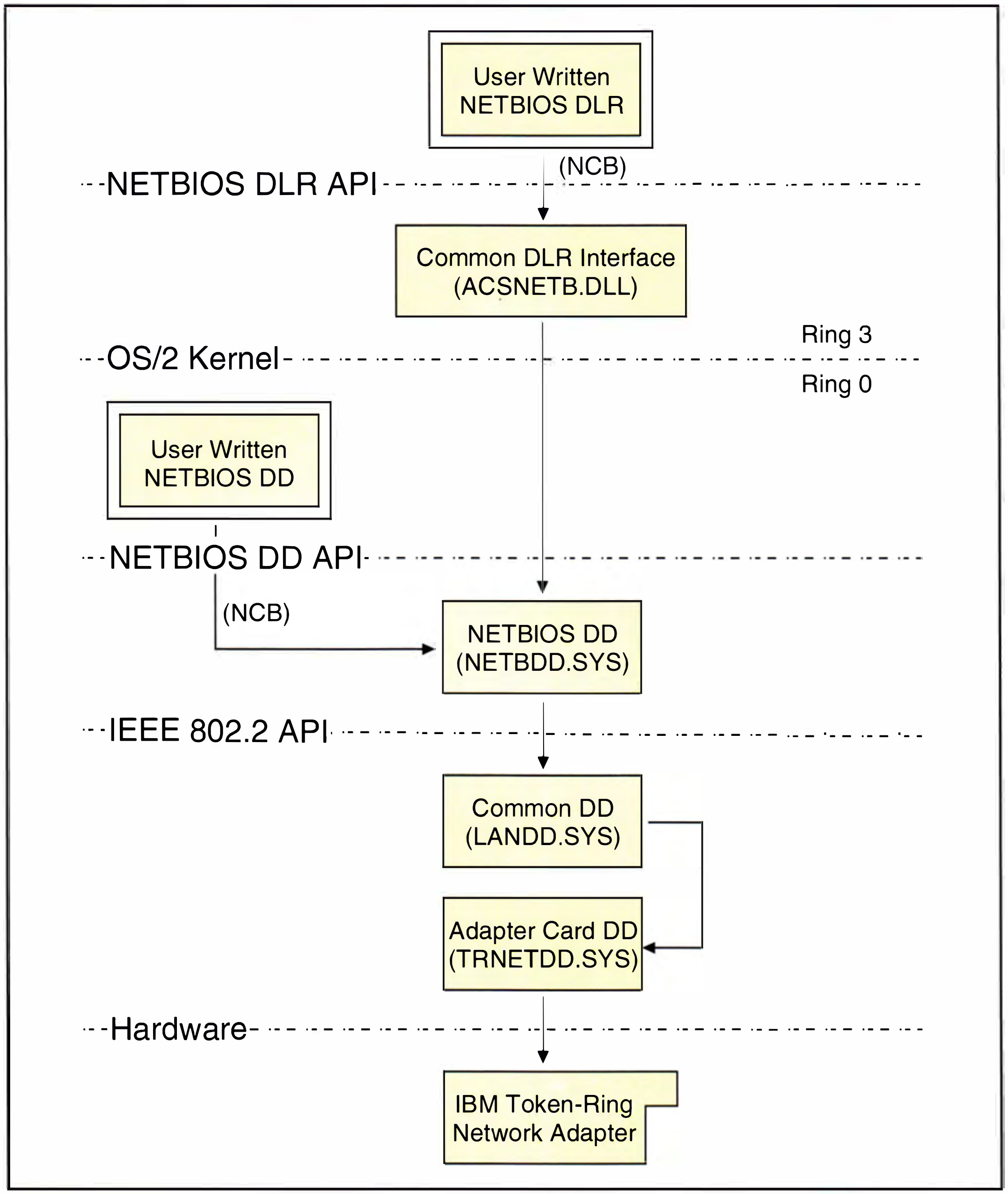


Figure 3. OS/2 EE NetBIOS Token-Ring API Flow

Because the invocation mechanism has changed, an application must FAR RETURN to NetBIOS instead of employing an INTERRUPT RETURN.

It is important to limit the number of data segments consumed for Receive/Transmit Buffers and NCB structures, since device drivers, including a user-written one, must lock the memory containing these structures.

Similarly, because adjustments cannot be made to segment boundaries, the length of the data plus its offset, cannot exceed 64K or the command will complete with a X'01' indicating and *illegal buffer length.*

The TRACE command is no longer supported. Instead, an OS/2 System level trace is provided.

INVOCATION METHODS

Figure 3 depicts the two programming methods for the NetBIOS API over Token-Ring. The procedures for *linking* to the API differ between the two interfaces, as follows.

Dynamic Link Routine Interface: A program written to the DLR interface must include the declaration of the NetBIOS Dynamic Link Routine module, ACSNETB, in its code. This declaration, along with other pertinent data structure declarations are provided with in the Communications Manager subdirectory CMLIB. Three language declarations are provided: C, PASCAL, and Assembly.

To runtime link to this DLR, an application program must first perform an OS/2 DosLoadModule on the DLR (ACSNETB). After the module has successfully loaded, a DosGetProcAddr is necessary to obtain the entry point for the DLR. Each process that run-time links to this module must free it via the DosFreeModule, before terminating its execution.

Load-time linking to the NetBIOS DLR is handled like any other external reference. A link record creates a hard coded reference to this module in the application's executable code. Once linking is complete, NCBs may be exchanged across this interface in a FAR CALL/RETURN fashion whereby the address of the NCB is pushed onto the stack before calling.

Device Driver Interface: A program written to the DD interface must first perform an ATTACHDD OS/2 DevHelp function to obtain the entry point of the IEEE 802.2 Device Driver (NetBIOS$), as well as to exchange the entry point and name of the application's device driver.

Upon successfully attaching to the NetBIOS Device Driver, the application device driver is free to issue NCBs across this interface in a FAR CALL/RETURN fashion. The ES:BX registers must point to the NCB and an invocation code of X'0000' must be pushed onto the stack prior to the call. Additionally, as a result of the architecture, all data control blocks and buffers accessed by the NetBIOS Device Drive must be in locked memory prior to passing them to NetBIOS on the call. It is the responsibility of the application device driver to perform this function.

API PROGRAMMING AIDS

OS/2 EE "CMLIB" subdirectory contains the following program data declaration include files to assist you in programming to the NetBIOS API:

NetBIOS Assembly Language Data Structures

```
NETB_1_A.INC : Command and Return
               Code Declarations
NETB_2_A.INC : Data Declarations
NETB_3_A.INC : Trace Data
               Declarations
NETB_4_A.INC : DLR Declaration
```

NetBIOS C Language Data Structures

```
NETB_1_C.H   : Command and Return
               Code Declarations
NETB_2_C.H   : Data Declarations
NETB_3_C.H   : Trace Data Declarations
NETB_4_C.H   : DLR Declaration
```

NetBIOS PASCAL Language Data Structures

```
NETB_1_P.INC : Command and Return
               Code Declarations
NETB_2_P.INC : Data Declarations
NETB_3_P.INC : Trace Data
               Declarations
NETB_4_P.INC : DLR Declaration
```

EVENT NOTIFICATION

Dynamic Link Routine Notification: Event notification for the DLR interface is compatible with PC DOS. The application has the option of issuing commands in a WAIT or NO-WAIT fashion. Commands issued with the WAIT option will return control to the application when the command has completed, along with a return code provided both in the AL register and the NCB_RETCODE field.

Commands issued as NO-WAIT allow the application to continue processing. If an appendage is provided, via the NCB_POST@ field of the NCB, upon command completion the application will be given control after the

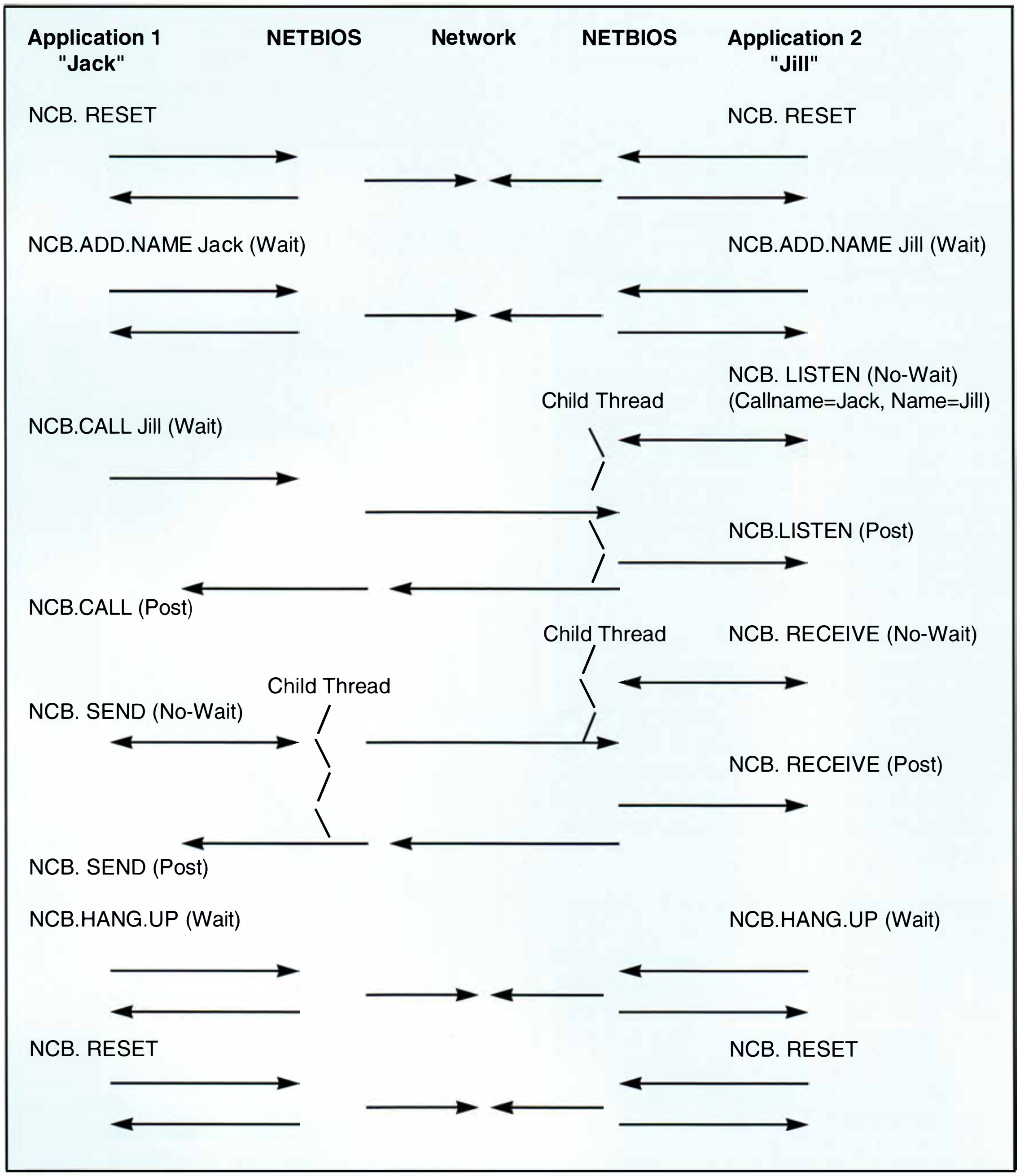


Figure 4. DLR Flow

final return code is placed in both the AL register and the NCB_RETCODE field. The application program would then continue its execution with the instruction pointed to by NCB_POST@.

If an appendage is not provided with a NO-WAIT command, the application program must check the NCB_CMD_CMPL field of the NCB to determine completion. An application should not poll the NCB_RETCODE field for completion, as the return code is set before the command actually completes. The command is complete when the NCB_CMD_CMPL field no longer contains an in-progress (X'FF') indicator.

Device Driver Notification: Event notification for Device Drivers is made through return codes and user supplied appendages. When an event completes, the NetBIOS Device Driver calls the entry point supplied by the application on the

ATTACHDD command. If the NCB_POST@ field is completed by the application prior to invoking the command, it indicates the point for the application to continue. The application appendage routine should be kept short, since NetBIOS is blocked until the application FAR RETURNs to NetBIOS.

DLR FLOW

Figure 4 illustrates a sample flow over the DLR API. They depict the steps taken to be able to establish a session, send one frame of data, and terminate communications.

This flow closely follows the sample program on the *IBM Local Area Network Technical Reference Sample Program Listings* diskette that comes with the *IBM Local Area Network Technical Reference.*

Both applications establish their operating environment by obtaining NetBIOS resources through an NCB.RESET. Each application then registers its name with the network.

Application 2 issues an NCB.LISTEN to permit a session to be established. As the LISTEN command was issued with the NO-WAIT option, NetBIOS creates a child thread to handle the command completion.

Application 1 calls *Jill* which results in a series of events: the NetBIOS thread waiting on the completion of the LISTEN is satisfied, it notifies application 2 of this event, and the thread is destroyed.

As the LISTEN is satisfied, a Local Session Number (LSN) is created for the session between Jill and Jack. Similarly, as the CALL completes, an LSN is returned signifying the session between Jack and Jill. Jill then issues a RECEIVE command for this LSN. Again, because the NO-WAIT option has been selected, NetBIOS creates a child thread to handle the completion of this event.

Meanwhile Jack, via Application 1, Sends data to Jill. The NetBIOS child thread waiting for data for Jill is satisfied. Another RECEIVE is not necessary, because the Receive Buffer provided with this NCB was large enough to contain the message Jack sent. Had the message Jack sent exceeded this buffer, Jill would need to issue subsequent RECEIVES to obtain the entire message.

Because Jack chose the NO-WAIT option when he sent his message, NetBIOS created a child thread to wait completion of this SEND. Since Jill received the entire message, and this is a session level SEND, an acknowledgment is sent back.[8] This acknowledgment satisfies the thread waiting notification of the outcome of the send, and Application 1 is notified of this event.

Both Jack and Jill terminate the session by issuing an NCB.HANG.UP. They then deallocate the NetBIOS resources they claimed earlier by issuing an NCB.RESET.

DD FLOW

Figure 5, the Device Driver Flow, illustrates appendage offsets in the NCBs. When the command completes, a final return code is posted in the AL register and the NCB_RETCODE field. The application program then continues execution at the appendage offset location (which remains in the completed NCB). Note, the appendage routines should be kept short because NetBIOS is blocked until the appendage has FAR RETURNed.

This shows the commands issued to establish a session, send one frame of data, then terminate communications. The important points to note are all commands are issued in a NO-WAIT fashion because the WAIT option is not available via the Device Driver interface.[9]

WHICH INTERFACE SHOULD YOU USE?

The NetBIOS DLR interface requires a simpler application program. A DLR program then uses semaphores and threads for event notification.

[8] *A return code of X'06' is posted in the NCB_RETCODE field if the receive buffers could not contain the entire message. Another RECEIVE can be issued to obtain the remainder of the message before an NCB.SEND timeout occurs.*

[9] With the exception of the *NCB.RESET* and the *NCB.CANCEL* commands.

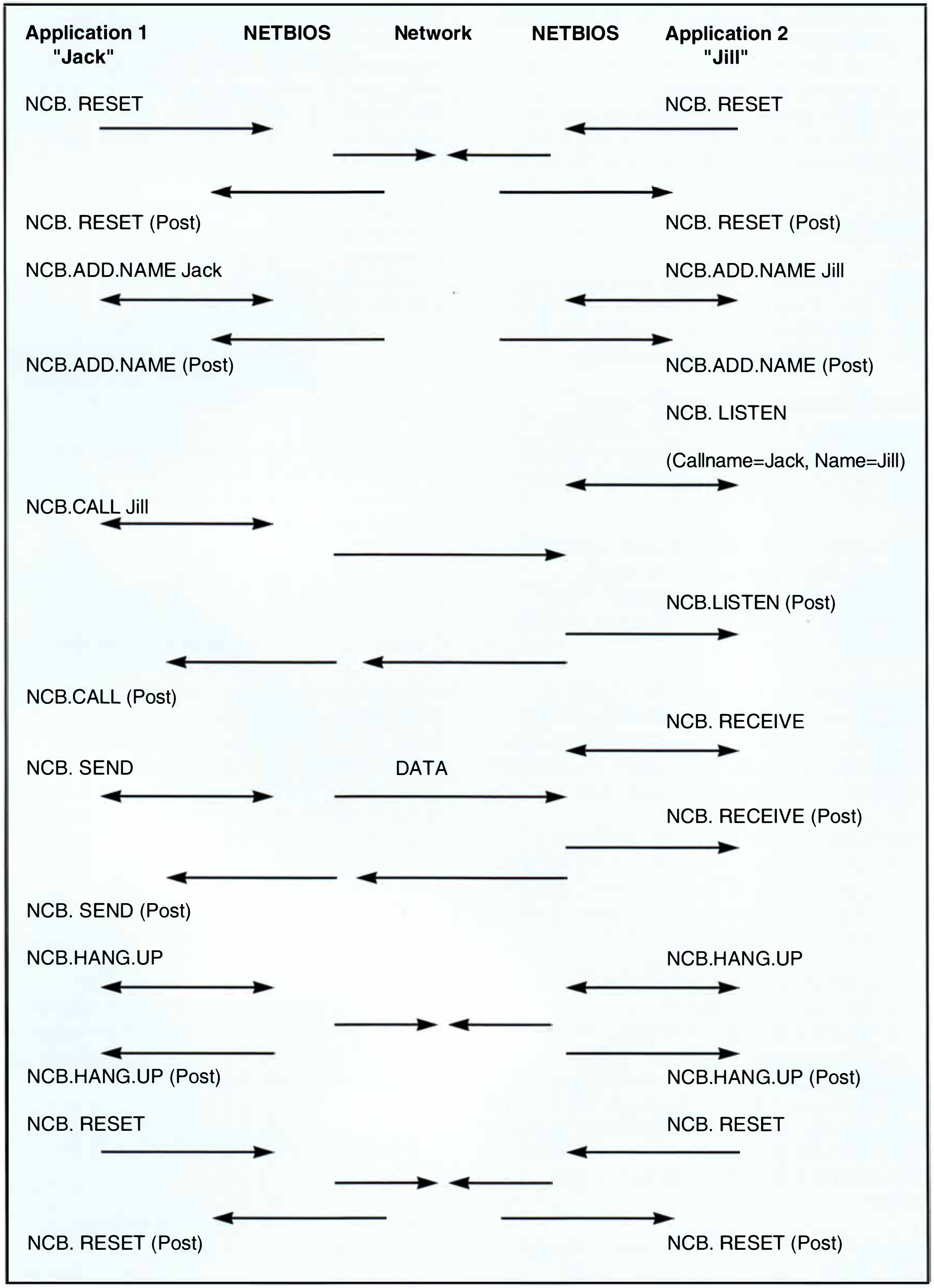


Figure 5. DD Flow

Next, a DLR application uses OS/2 IOCTL calls to exchange data between itself and the NetBIOS Device Driver. This overhead results in a performance reduction because a DLR application and a device driver execute at different privilege levels.

On the other hand, the NetBIOS Device Driver interface requires the interfacing application to be a device driver. Thus, a more complex user application is required. However, performance gains result because event notification is a FAR CALL/RETURN between two applications executing at the same privilege level.

OS/2 EE CONFIGURATION FOR THE IEEE 802.2 API

Two forms of configuration are required to run applications over the NetBIOS API: CONFIG.SYS and the Communications Manager configuration file specified on the NetBIOS DEV= statement in CONFIG.SYS.

The OS/2 NetBIOS API software is loaded via CONFIG.SYS. Depending on which IEEE 802.2 interface your application uses, different statements must be included in the CONFIG.SYS file.

The NetBIOS DLR interface requires the NetBIOS Device Driver which interfaces to a common IEEE 802.2 Device Driver, which in turn communicates with the adapter card Device Driver. Below is a sample CONFIG.SYS for this interface.

```
DEV=C:\CMLIB\LANDD.SYS
DEV=C:\CMLIB\TRNETDD.SYS  CFG=C:\CMLIB\your.CFG
DEV=C:\CMLIB\NETBDD.SYS   CFG=C:\CMLIB\your.CFG
RUN=C:\CMLIB\ACSTRSYS.EXE
```

The NetBIOS Device Driver interface also requires the IEEE 802.2 Common Device Driver and the adapter card device driver. Below is a sample CONFIG.SYS for this interface:

```
DEV=C:\CMLIB\LANDD.SYS
DEV=C:\CMLIB\TRNETDD.SYS  CFG=C:\CMLIB\your.CFG
DEV=C:\CMLIB\NETBDD.SYS   CFG=C:\CMLIB\your.CFG
```

The ordering of these device drivers is important. LANDD.SYS must be the first LAN Device Driver installed, followed by the Device Driver support for the adapter type. NetBIOS is the last Device Driver loaded.

During Device Driver load time, open and initialization messages are written to a log file called ACSLAN.LOG. This file is written to the same Drive:\Path that appears on the CFG= parameter of the DEVICE= statement. Figure 6 shows the sample listing of ACSLAN.LOG.

The second form of configuration is through Communications Manager Profiles. Since NetBIOS uses the IEEE 802.2 interface, it is necessary that the IEEE 802.2 profile and the NetBIOS profile not conflict.

The IEEE 802.2 profile sets the overall adapter card operating environment for all applications using this adapter. The NetBIOS profile defines the overall resource pool for all NetBIOS applications employing this adapter. Both the IEEE 802.2 profile and the NetBIOS profile reside under *LAN*

Successful Open/Initialization:

```
Accessing IBM Token-Ring Local Area Network. Please wait.
Adapter 0 is initializing.
Adapter 0 is set for a data rate of 4 MB/s.
Adapter 0 opening.
Adapter 0 initialized and opened successfully.
Adapter 0 is using node address 400000011952.
Adapter 0 has an adapter microcode level of A58769B.
IBM NETBIOS Version 3.0 is loaded and operational.
```

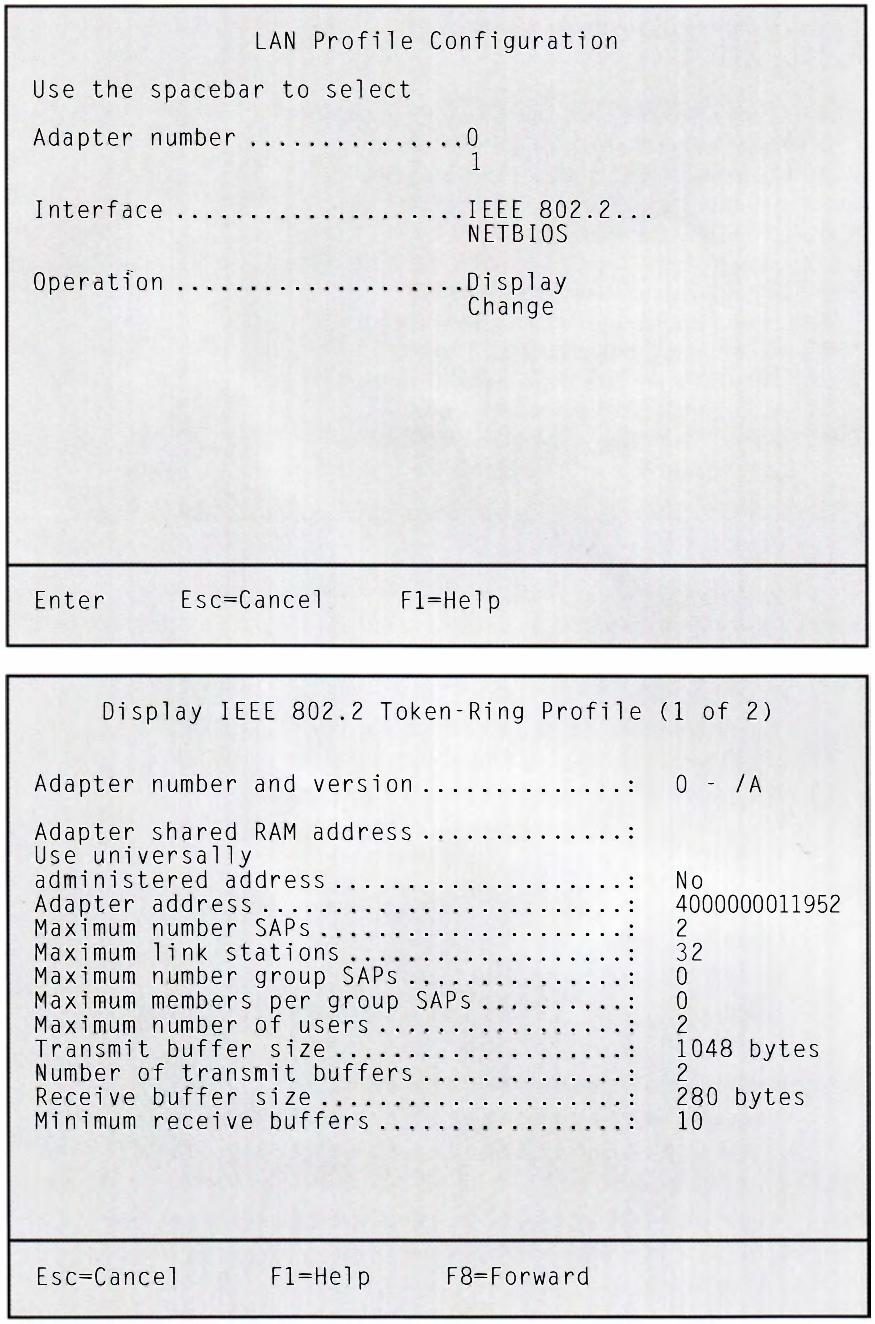
Unsuccessful Open/Initialization:[10]

```
Accessing IBM Token-Ring Local Area Network. Please wait.
Adapter 0 is initializing.
Adapter 0 is set for a data rate of 4 MB/s.
Adapter 0 opening.
Adapter 0 initialized and opened successfully.
Adapter 0 is using node address 400000011952.
Adapter 0 has an adapter microcode level of A58769B.
ACS0531: A configuration file has not been specified on the NETBIOS
DEVICE= statement.
ACS0530: A NETBIOS error occurred. For additional information,
access the ACSLAN.LOG file. Press any key to continue.
IBM NETBIOS Version 3.0 is loaded and operational.
```

Figure 6. A Sample Listing of ACSLAN.LOG

[10] Error messages written to this log file are documented in the *IBM Operating System/2 Extended Edition Problem Determination Guide for the Service Coordinator (90X7778).*

Feature Profiles. Figure 7 shows the IEEE 802.2 profile screen displays, and Figure 8 shows the NetBIOS screen.



```
                  LAN Profile Configuration

 Use the spacebar to select

 Adapter number ...............0
                               1

 Interface ....................IEEE 802.2...
                               NETBIOS

 Operation ....................Display
                               Change

 Enter     Esc=Cancel     F1=Help
```

```
      Display IEEE 802.2 Token-Ring Profile (1 of 2)

 Adapter number and version .............:   0 - /A

 Adapter shared RAM address .............:
 Use universally
 administered address ...................:   No
 Adapter address ........................:   4000000011952
 Maximum number SAPs ....................:   2
 Maximum link stations ..................:   32
 Maximum number group SAPs ..............:   0
 Maximum members per group SAPs .........:   0
 Maximum number of users ................:   2
 Transmit buffer size ...................:   1048 bytes
 Number of transmit buffers .............:   2
 Receive buffer size ....................:   280 bytes
 Minimum receive buffers ................:   10

 Esc=Cancel      F1=Help     F8=Forward
```

Figure 7. LAN Profile Screens

The important points to grasp here are:

1. Through the IEEE 802.2 profile, the overall operating environment for the adapter card is defined. Each application claims a portion of the overall adapter resources when it performs a logical open of the adapter (DIR.OPEN.ADAPTER for APPC, and NetBIOS Device Driver at Open/Initialization time). If users are simultaneously using an adapter, then the sum total of their claimed adapter resources cannot exceed the values set for this profile.

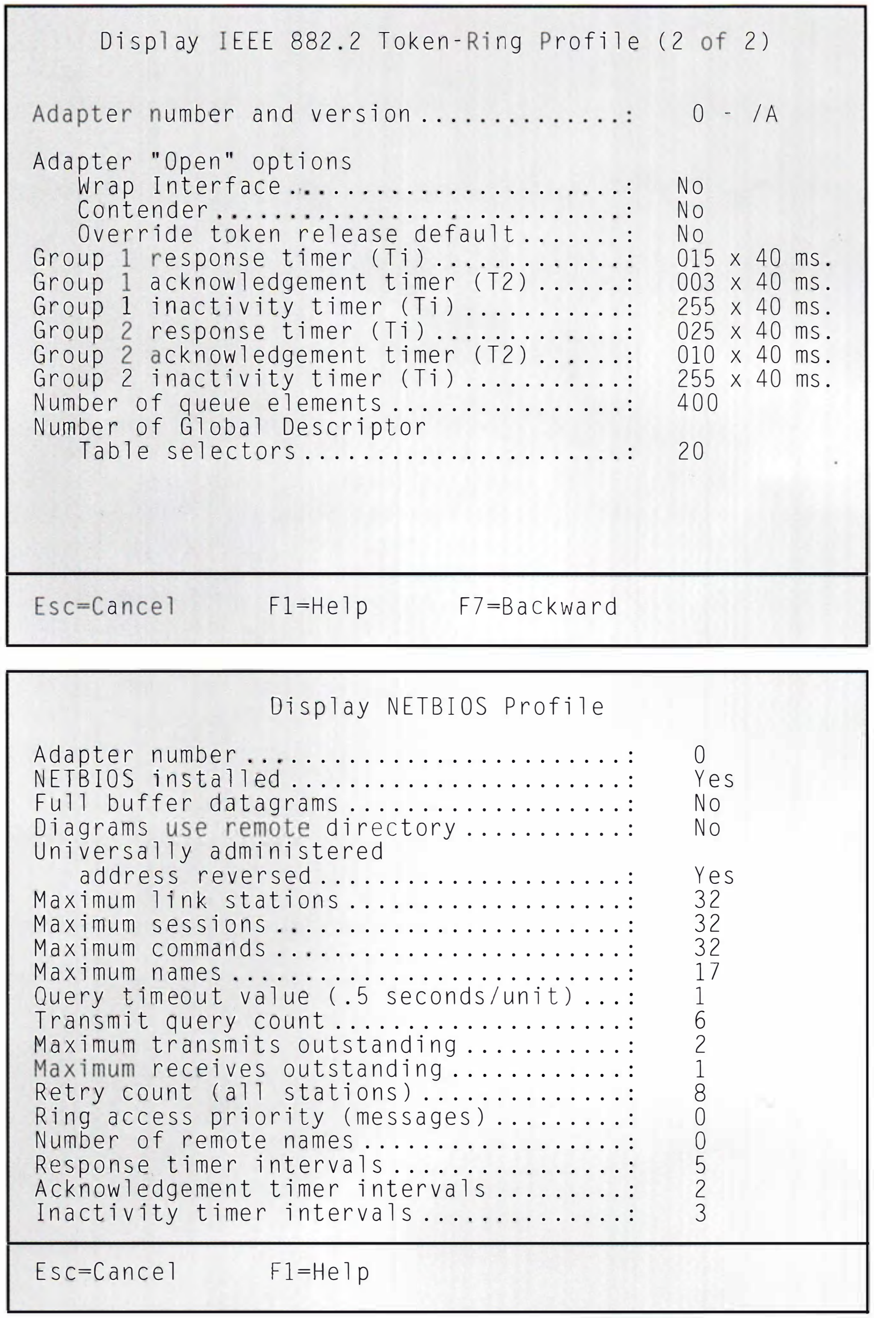
2. The overall NetBIOS resource pool is defined through the NetBIOS profile. The sum total of simultaneously active NetBIOS applications requesting resources via an NCB.RESET cannot exceed the values defined for this profile.

SUMMARY

The structure of the NetBIOS API is largely a result of OS/2 architecture, which was designed to maintain compatibility with PC DOS while providing multi-process, multi-threaded function. A standard Network Control Block interface is provided as the means to accessing a network regardless of media type.

In addition to providing a standard Network Control Block interface, the NetBIOS API offers flexibility to write either a Dynamic Link Routine or Device Driver application.

By understanding general command flows over each interface, invocation methods, and the configuration requirements, writing an application to the NetBIOS API is facilitated.



```
        Display IEEE 882.2 Token-Ring Profile (2 of 2)

Adapter number and version..............:     0 - /A

Adapter "Open" options
   Wrap Interface........................:   No
   Contender.............................:   No
   Override token release default.......:   No
Group 1 response timer (Ti).............:   015 x 40 ms.
Group 1 acknowledgement timer (T2)......:   003 x 40 ms.
Group 1 inactivity timer (Ti)...........:   255 x 40 ms.
Group 2 response timer (Ti).............:   025 x 40 ms.
Group 2 acknowledgement timer (T2)......:   010 x 40 ms.
Group 2 inactivity timer (Ti)...........:   255 x 40 ms.
Number of queue elements................:   400
Number of Global Descriptor
   Table selectors......................:   20

Esc=Cancel       F1=Help       F7=Backward
```

```
                  Display NETBIOS Profile

Adapter number..........................:     0
NETBIOS installed.......................:     Yes
Full buffer datagrams...................:     No
Diagrams use remote directory...........:     No
Universally administered
   address reversed.....................:     Yes
Maximum link stations...................:     32
Maximum sessions........................:     32
Maximum commands........................:     32
Maximum names...........................:     17
Query timeout value (.5 seconds/unit)...:     1
Transmit query count....................:     6
Maximum transmits outstanding...........:     2
Maximum receives outstanding............:     1
Retry count (all stations)..............:     8
Ring access priority (messages).........:     0
Number of remote names..................:     0
Response timer intervals................:     5
Acknowledgement timer intervals.........:     2
Inactivity timer intervals..............:     3

Esc=Cancel       F1=Help
```

Figure 8. LAN Profile Screens

REFERENCES

IBM. *LAN Technical Reference.* (SC30-3383).

IBM. *NetBIOS Application Development Guide.* (S68X-2270).

IBM. *Operating System/2 Extended Edition Problem Determination Guide for the Service Coordinator.* (90X7778).

IBM. *Operating System/2 Programming Guide.* (6280211).

Dana L. Beatty, *IBM Entry Systems Division, 1000 N.W. 51st Street, Boca Raton, Florida 33429. Ms. Beatty is a staff programmer in the Communications Subsystem Development I department. She joined IBM in 1985 in Austin as a systems analyst and transferred to the Local Area Network group for OS/2 EE Communications Software Products in 1986. There, she participated in the design and development of applications using the IEEE 802.2 API. Ms. Beatty received a BA in Computer Science from the University of Texas in Austin.*

LAN Requester and Server

OS/2 LAN Server Command Line Interface

Carolyn Easter

Roy Feigel

by Carolyn Easter and Roy Feigel

Operating systems provide commands to control specific functions, view information, or configure systems. Both DOS and OS/2 provide a rich set of commands for its users as do network operating systems. The OS/2 LAN Server and PC LAN Program include a command line interface as well as a full screen interface. However, many advanced users prefer the single command interface. In order to provide an improved set of commands, this interface for the IBM OS/2 LAN Server has been greatly enhanced.

The IBM OS/2 LAN Server versions 1.2 and 1.3 provide a command line interface consistent with IBM PC LAN Program Version 1.3x, the DOS LAN requester command line interface, and include advanced functions for administration. This interface can now be used as an alternative to the full screen interface for most administration and management functions. The commands can be categorized by function:

- Startup, Shutdown
- Sharing and Using Resources
- Managing Printers and Serial Devices
- Messaging
- Utilities
- Administration and Information

The administration commands comprise the majority of the new functions for the command line interface. The other enhancements expand the IBM single system's image concept to commands previously enabled only through the full screen interface and provide extended help information. This article is written primarily to provide information about the new administrative command line commands.

Some functions provided by the command line interface are not available through the full screen interface and vice versa. For example, to change the password expiration period, the NET ACCOUNTS command must be issued. To change or add logon assignments, the administrator must use the full screen interface. Figure 1 lists the functions available only through the command line.

Users or administrators can create useful utilities using the REXX interpretive language which is packaged with the OS/2 Extended Edition 1.2 and with OS/2 Extended Edition 1.3 and OS/2 Standard Edition 1.3. The REXX programs are command (batch) files with the .CMD extension. These batch files can execute OS/2, LAN Server, SQL Statements and REXX commands.

Tools and customized applications can also be created using the OS/2 LAN Server Application Programming Interface. This interface is extremely flexible and contains a higher degree of function. However it does require knowledge of programming techniques in the OS/2 Operating System environment and will not be part of this article. The Command Line Interface is fully documented in the *OS/2 LAN Server Command Line Reference*, order number S33F-9431-0.

NEW ADMINISTRATIVE COMMANDS

The new administrative commands are those that can permanently change the configuration of a server. These commands provide the following functions:

Setup and manage user IDs: NET USER
Setup and manage group IDs: NET GROUP
Define resources: NET ALIAS
Establish access control: NET ACCESS
Configure NETLOGON Service: NET ACCOUNTS

SETUP AND MANAGE USER IDS – NET USER

The NET USER command allows an administrator to view the defined user list; view a specific user; modify, create or delete a user ID.

When the administrator creates a user ID either from the User Profile Management interface or through the command line interface, the new user ID and password (and other user specific information) is recorded in a file named NET.ACC on the server designated as the domain controller.[1] This file contains all user IDs, group IDs, and the access control information for resources on that machine. It is checked periodically for changes by the NETLOGON service. When changes are detected, the changed information is extracted and routed to all servers that are defined and started in the domain. Each server in the domain, therefore, maintains the information about all defined users in the domain and all groups in the domain but keeps the access control information only about its own resources.

IBM OS/2 LAN Server also provides a feature for creating unique logon environments for each user. At logon, a user can be provided applications, and connected to network drives and printers. This unique environment is setup by an administrator through the full screen interface for LAN Server. These logon assignments are stored in unique subdirectories on the domain controller in specially formatted files. These files and other data are part of what is known as the domain control data base, stored on the domain controller server in the subdirectory d:\IBMLAN\DCDB\USERS\name where *d:* is the drive letter on which the LAN Server code is installed and *name* is the name of the user ID. When the user logs on, the LAN Requester retrieves the stored profile for this user ID, extracts the profile information, and performs the connections for the user. The selected applications available for the user are dynamically added to Presentation Manager Desktop (or DOS LAN Requester Served Applications panel).

Command Line only functions

Command	**Function**
NET ACCESS	• Setup auditing for specific failures (e.g. audit on fail to write)
NET ACCOUNTS	• All functions
NET USER(S)	• Specify the times of day and days of the week for valid logon • Specify valid workstation

Figure 1. Functions Available Exclusively from the Command Line

Home directories currently are established on any server in the domain and are created under the same directory for the LAN Server code. The directory structure is d:\IBMLAN\USERS\name where d: is the drive letter on which the LAN Server code is installed and *name* is the name of the user ID.

[1] Note that this is true for administrators logged on to a *LAN Server* either locally at the domain controller server or at a remote workstation. If the administrator is logged on locally at a workstation (not logged on to a *LAN Server*), any created *user ID* would go into the local *NET.ACC* file stored in the *\MUGLIB\ACCOUNTS* subdirectory for *non-LAN requesters* and in *IBMLAN\ACCOUNTS* subdirectory for *LAN requesters*.

With the NET USER command an administrator can list, create, modify, or delete user IDs. In fact, user IDs can be created on a requester machine as well as a server. The local IDs are established for use of a local database, for local logon, and to logon then attach to a host. For local area network use, local IDs are not necessary.

In the following examples, it is assumed that the commands are issued while logged on at the domain controller and therefore, the NET ADMIN command prefix is not needed.

```
NET USER CAROL a4532 /ACTIVE:no
/PRIV:user /ADD
```

In this example, the user ID Carol will be added with privilege level of user, password of a4532, and will not be allowed immediately to logon based on /ACTIVE:no. This will allow the user ID to be further setup through the LAN full screen for logon assignments.

```
NET USER CAROL
/WORKSTATIONS:B2FL8012 /TIMES:
8:00-5:00,M-F
```

The user CAROL account has been changed to reflect logon times and workstations allowed for logon.

The workstation names allowed for each user and the logon times allowed are functions provided by the command line only. The workstation name for a requester is an eight-character name specified in the IBMLAN.INI file for OS/2 workstations and in the DOSLAN.INI file for DOS LAN Requester workstations. The user may change this name simply by editing the appropriate file or by issuing the NET START command and specifying the name of the workstation as a parameter. This would imply that if the user knew the list of workstation names allowed for his user ID then the user could logon at any workstation, changing the name to one recognized in the user's profile list.

In the case where the logon times are established, the user may log on starting at 8:00 (assumed to be a.m. since the time is in 12 hour format) up until 5:00 PM (assumed to be PM since it follows 8:00 and must be a later time. Twenty-four hour format may also be used.). If the user is logged on past 5:00, the server takes no action unless other parameters are specified for server machines using the NET ACCOUNTS command.

SETUP AND MANAGE GROUP IDS – NET GROUP

The NET GROUP command allows an administrator to list the groups defined in the domain, list a specific group, or add, delete, and change group membership. The NET GROUP command can also be issued as NET GROUPS. When a group ID is created, the name of the group and the user ID names are added to the NET.ACC file on the domain controller. These names are then propagated to all servers in the domain.

To create a group ID from the command line the administrator simply types the NET GROUP command then the name of the group specifying /ADD function. Group names must be created before user ID names can be added.

```
NET GROUP DESIGN /ADD
/COMMENT:"Design group"
```

In this example the group name DESIGN is a group name created with the comment *Design group*. The administrator can now issue a command to add users to the group.

```
NET GROUP DESIGN CAROL ROY AL /ADD
```

The group DESIGN now has members CAROL, ROY, and AL.

DEFINE RESOURCES – NET ALIAS

The NET ALIAS command allows a user or an administrator to browse the list of aliases defined in the domain and to connect to a resource defined by the alias with no need to be aware of the location of the resource. Aliases are nicknames for resources defined in LAN Server domains. They are the exclusive naming service for the IBM LAN Server product.

The administrator may use the NET ALIAS command to create, delete, or change the definition of an alias. From the full screen interface, the administrator may select an alias and change the access control rights for the resource. When using the full screen interface, defining aliases and access control is managed by a single set of panels. The creation of an alias from the command line is done by typing the NET ALIAS command with the name of the server and the name of the resource as part of the command. The access control specifications must then be accomplished by issuing the NET ACCESS command preceded by the NET ADMIN command to the server defined by the alias.

The NET ALIAS command is the only command which updates the domain control database without having to issue the NET ADMIN command or specify whether the alias definition is to be added or changed. The/ADD switch is not necessary when adding an alias and the /DELETE command is necessary only when deleting aliases.

In addition to the exclusive nature of the NET ALIAS function, the administrator can be assured that no user may view an alias name unless the user has some access rights to that alias. Whether the user selects Actions and Network Assignments from the full screen interface or issues the NET ALIAS command from the command line, the only list of aliases presented are the names of resources to which the user has at least read access.

When an administrator wishes to change the location of an alias, they delete the alias and recreate it using the new specification for the resource. When the user connects to this resource either by selecting an application or issuing a NET USE command, the connection is made to the new resource location and the user environment appears to remain constant.[2]

A user wishing to connect to a named resource can issue the NET USE command and specify the resource with an alias name. This gives a twofold advantage: First, they may list the alias names and decide which name represents the resource needed. (This assumes the administrator has created aliases with appropriate and recognizable names.) The user may then type the NET USE d: ALIAS command where *d:* is the local drive letter to use for the connection and ALIAS is the name of the resource. The second benefit is efficient use of share names at the server. The alias may not be shared at the server where defined, but the NET USE action causes the share to occur and makes the connection to the resource simultaneously with one command.

Alias definitions are also used when defining applications to the LAN Server. An administrator may define an application as one of three types: OS/2 Public Application, DOS Public Application, or Private Application. These applications can be defined using alias names. OS/2 Public Applications and DOS Public Applications appear dynamically when the user logs on. An OS/2 Requester user will have the OS/2 Public Applications appear in a dynamically created group called Public Applications and the DOS LAN Requester user will have the applications appear on the DOS LAN Requester Served Applications menu. When an application is selected from these interfaces, the resource selected by the alias location will be dynamically shared, the connection made, and the application started at the user's request.

In the example:

```
NET ALIAS BOOKS \\IBMSRV C:\BOOKSCR
/WHEN:REQUESTED
```

- The alias name will be created for the named server/resource if not defined
- The directory will be created if not already created
- The alias definition will be changed if the alias exists

The command:

```
NET ALIAS BOOKS /D
```

will delete the alias name BOOKS.

[2] If a resource is *NET SHARED* by the alias name and the alias is deleted and recreated defined as a new location, the share is not affected. Therefore, if the administrator wishes for all subsequent connections defined by this alias name to be to a new location, the administrator must delete the share. Subsequent connections to the new alias definition will now automatically share the new resource and the connection will be made to the new location.

Other parameters may be of significant value to administrating resources. To prevent an application from being executed by more than the allowed number of users (as specified by the license agreement), the administrator can specify the maximum number of concurrent users for a resource. This is done by the /USERS:nn parameter when changing or creating the alias. The same specification can be done through the NET SHARE command provided the resource is shared at the time of specification. The NET ALIAS command is a way of storing the specification before the share is done.

For serial device queues and print queues, the command line switch must be specified when creating aliases. This is done automatically when the alias is created from the full screen interface.

ESTABLISH ACCESS CONTROL – NET ACCESS

The NET ACCESS command allows an administrator to list, change, create, or delete access control information about a resource on a server. Each server maintains its access control profiles and lists. The access control profile is an entry in the NET.ACC file on the server in the IBMLAN\ACCOUNTS directory. The access control list details the users, groups and access rights for each.

The access control list may have 64 entries for any access control profile. For domains with many users defined, it is most efficient to define groups and associate the access control information with them. In fact, the purpose of groups is to make access control lists concise.

If an administrator sets auditing of a resource to *all accesses* an audit log entry will be generated for each access. If this resource is used often, it could impact the performance of the server and fill the audit log quickly. Setting the auditing for a resource as *failed accesses only* may be sufficient for monitoring. More specific settings for auditing can be established by using the command NET ACCESS. For example, the auditing can be set to *failed access* when attempting to create a file in a directory. Failure to create the file will then generate an audit log entry but other access failures will not generate an audit log entry. To create an address control profile, the /ADD parameter is required. An *empty* access control profile can also be created. In this case, the profile exists but no permissions are specified.

In the example:

```
NET ACCESS C:\DW4DOC /ADD
SALESGR:CDRW
```

the access profile for the directory is created and has one access control list element added. This entry allows the SALESGR ID create, delete, read and write access to the directory. The SALESGR may be a group or a user ID.

```
NET ACCESS C:\DW4DOC /GRANT ROY:RW
```

If the access control profile already exists for the directory, the /GRANT switch adds an entry in the access control list for the user ID ROY with privileges to read and write to the directory.

```
NET ACCESS C:\DW4DOC /TRAIL:YES
```

The /TRAIL switch turns on auditing of all activity to the resource. With the above code, the audit trail is turned on for this resource. If the auditing is not turned on generally for the server as specified in the IBMLAN.INI file, the auditing will not be done.

```
NET ACCESS D:\EMPLOYEE\DIRECTORY.LST
/FAILURE:WRITE
```

Here, only write failures to file DIRECTORY.LST file are audited. Note that this is an example of an access control profile for a single file instead of a directory. The directory can have one set of specifications and each file in the directory can have individual specifications for access control and auditing.

CONFIGURE NETLOGON SERVICE– NET ACCOUNTS

The NET ACCOUNTS command changes or views the behavior of NETLOGON, the service that manages the logon and session connection for users and runs at the domain controller and additional servers. If a user is defined in the domain then a connection to a shared resource or alias is assured, since the user has already been pre-validated by the logon service. If the account is expired the connection would fail.

NETLOGON defines several functional criteria for LAN Server operations:

- Server/workstation type
- Password management
- Connection/reconnection capabilities

The servers in a LAN Server domain must be one of three types:

- Domain controller server
- Member status server
- Backup status server

Workstations are simply STANDALONE status.

The status of a server or workstation can be viewed by typing NET ACCOUNTS. The role of the workstation is displayed along with:

- Account expiration grace period
- Minimum password age
- Maximum password age
- Minimum password length
- Unique password history

The following examples show how an administrator can set or change the length of time in days that a user's password is valid. The administrator must stop the NETLOGON service first by using the NET STOP NETLOGON command, then restart the service for the changes to take effect. The service is started by issuing the NET START NETLOGON command.

```
NET ACCOUNTS /MAXPWAGE:30
```

Sets the password expiration period to 30 days. After a user changes their password the expiration date is recalculated. This date for each user can be viewed by typing the NET USER command.

```
NET ACCOUNTS /MINPWAGE:30
/MAXPWAGE:45
```

This command enforces the unique use of passwords over a period of at least eight months. After a user changes their password, they may not change it again until 30 days has passed and must change it before 45 days has passed. The unique password history setting for LAN Server is set to eight by default therefore a user in this example must use eight unique passwords before reusing one.

```
NET ACCOUNTS /FORCELOGOFF:10
```

In this example, when a user account expires, the server will disconnect the user session and not allow the user to reconnect. The specification on the account expiration grace period is a length of time in minutes and allows continued connection until the grace period has expired. During this time messages in the form of alerts are generated by the server to the user and are sent multiple times over the grace period length (10 minutes) to ensure that the user is notified.

The user account expires if the TIMES parameter is specified for an individual user ID and comparison with the local time indicates that the user account has expired. Another parameter in the user profile specifies a period of time or date of expiration. This will also cause a user ID account to expire.

This behavior for the NETLOGON service can be specified for each server.

SINGLE SYSTEMS IMAGE

In order to simplify administration and use of local area network resources, IBM utilized the concept of grouping servers into what is called a domain. IBM OS/2 LAN Server incorporated this concept into the full screen interface for users and administrators so that the resources can be used and managed from a single workstation and the users can be shielded from the complexity of the system. The OS/2 LAN Server 1.2 command line interface now expresses some of the single systems image concepts that had previously been available only from the full screen interface.

One of the most important changes to the command line is the addition of the NET ALIAS command which creates, displays, and changes the description of a resource. The alias name can then be used to define applications, logon assignments, and make connections to resources. There is a double benefit when using the alias name to connect to a resource.

The usual way to connect to a resource is to use the NET USE command.

```
NET USE D: \\DOM01\APPS
```

This command will connect the D: drive letter to the server resource as specified by the \\*server**netname* syntax. For the connection to be made, the netname APPS must be already shared at the server DOM01.

The single systems image concept, however, makes this command much easier both on the user of the command and on the administrator of the system. Assume that WRITEAPP has been defined as an alias and that the alias points to the server DOM01 and to the directory C:APPLS. The definition of the alias describes the resource location, when it will be shared, the number of users who can use it concurrently, and its comment string.

```
NET USE D: WRITEAPP
```

When the user issues this command, the administrator does not have to NET SHARE this resource for the users to connect and use it. There are other changes to the command line that incorporate the domain name as a parameter or switch for defining their realm of operation:

- **NET SEND:** allows a domain name when sending messages so that the broadcast is made only to the machines where the domain name is being used.
- **NET WHO:** asks the question *who's logged on* to the default domain. For administrators, the NET WHO command can query a specific domain.
- **NET PASSWORD:** can be used by users and administrators for managing multiple domain logon passwords.
- **NET TIME:** sets the local workstation time based on a domain controller server time.
- **LOGON:** allows a user to specify the name of a domain other than the default domain as specified in the IBMLAN.INI file or the one specified on the NET START command.
- **NET USE:** is probably the most used command in the network. It is used to make a connection from a local device (usually a drive letter like p: or a printer device such as lpt3:) to a remote resource on a server machine.

For administrators interested in handling multiple domains, the OTHDOMAINS parameter in the IBMLAN.INI file allows other server groups to be viewed with The NET VIEW command. The domain names specified are queried at requester startup and added as NetBIOS group names for the local workstation.

SUMMARY

The OS/2 LAN Server command line interface offers a greatly improved administration capability by providing specific commands to add local area network objects to the network configuration. These commands, used in combination with Operating System/2 commands, provide an alternative to the LAN Server full screen interface for users and administrators. Batch command files created either in the normal format or the REXX format provide automation for configuration and management of networks and systems. With just a little imagination and creative instincts, users and administrators of OS/2 LAN Server networks can provide tools and applications for their customers.

Carolyn Easter, *IBM Corporation, Internal Zip 3106, 11400 Burnet Road, Austin TX 78758, began working with IBM in 1980. Now an advisory programmer with experience in technical support, programming, and marketing support, she works in the Austin PS Programming Center in LAN System Products Planning. She provides technical marketing support which includes presentations for the Field Television Network, Developer Seminars, and field education. Mrs. Easter received a BS in Mathematics from Southern Methodist University and a BA in Computer Science from the University of Texas.*

Roy Feigel, *IBM Corporation, Internal Zip 3107, 11400 Burnet Road, Austin TX 78758, is an advisory programmer. In 1982, Mr. Feigel joined IBM as a programmer assigned to text applications for the 5520 Administrative System. In 1985 he became responsible for the server component of the PC LAN Program 1.1 and has since been involved in design and development of LAN products. Mr. Feigel is currently assigned to the System Design and Architecture group. He was the lead designer for the OS/2 LAN Server 1.2, Extended Edition 1.2 LAN Requester and DOS LAN Requester and has been a presenter for IBM's Technical Coordinator Program television broadcasts and an advisor for customers' large network designs. He received a BS in computer science from the University of Southwestern Louisiana in 1982.*

Extended Edition Database Manager

Database Application Remote Interface Primer: You Too Can Feel the Power!

Dwayne C. Jacobs

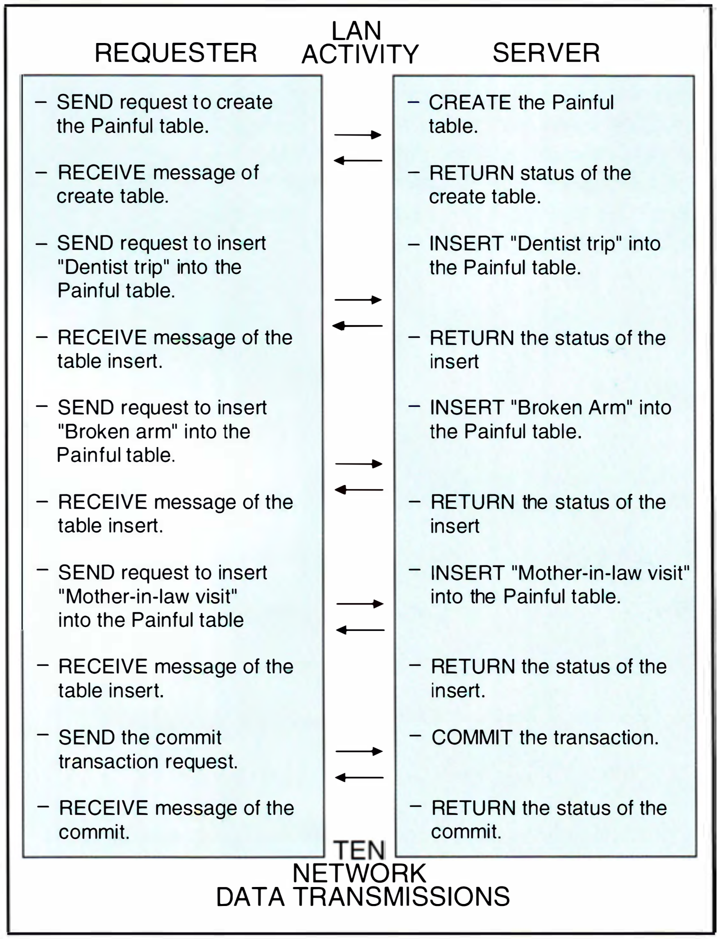
by Dwayne C. Jacobs

This is one of two articles which appear in this issue of the Developer that discuss the OS/2 EE Database Manager's Database Application Remote Interface (DARI). This article can be considered an overview of DARI, while the next article discusses a specific programming technique using the interface "Avoid the Network Traffic Jam: Passing Blocks of Data Using DARI".

Have the network blues got you down? Are your distributed database transactions taking a lifetime to complete? Does this make using your computer system a *painful* experience? Well, has OS/2 EE Database Manager got the solution for you!

In the database application environment there are many situations which are repetitive; receiving a fixed set of inputs, performing the same multiple requests of a database, and returning a fixed set of outputs. Typically this is accomplished using several SQL (Structured Query Language) statements. Each SQL statement, in the distributed environment, would require at least two data transmissions over a Local Area Network (LAN). If the transaction (the group of related SQL statements) is comprised of many SQL statements, it is obvious that this would result in considerable network traffic.

Furthermore, if many nodes are simultaneously executing the same transaction the network activity would be multiplied. This concurrent activity would result in a performance degradation for the entire LAN. See Figure 1 for a depiction of typical network traffic for a simple transaction. Notice that each SQL statement requires two network transmissions.



REQUESTER	LAN ACTIVITY	SERVER
– SEND request to create the Painful table.	→	– CREATE the Painful table.
– RECEIVE message of create table.	←	– RETURN status of the create table.
– SEND request to insert "Dentist trip" into the Painful table.	→	– INSERT "Dentist trip" into the Painful table.
– RECEIVE message of the table insert.	←	– RETURN the status of the insert
– SEND request to insert "Broken arm" into the Painful table.	→	– INSERT "Broken Arm" into the Painful table.
– RECEIVE message of the table insert.	←	– RETURN the status of the insert
– SEND request to insert "Mother-in-law visit" into the Painful table	→	– INSERT "Mother-in-law visit" into the Painful table.
– RECEIVE message of the table insert.	←	– RETURN the status of the insert.
– SEND the commit transaction request.	→	– COMMIT the transaction.
– RECEIVE message of the commit.	←	– RETURN the status of the commit.

TEN NETWORK DATA TRANSMISSIONS

Figure 1. Typical Network Data Transmissions

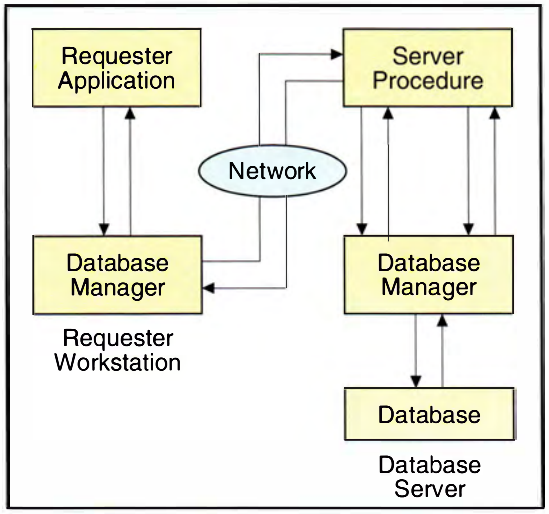


Figure 2. Execute a User Generated Stored Function Against a Specific Database

To support the user in a situation where performance is critical, OS/2 EE Database Manager (versions 1.2 and higher) provides the Database Application Remote Interface (DARI). This facility furnishes the means to execute a user generated *stored function* against a specified database. A stored function is a program which is created by the user and is built into a Dynamic Link Library (DLL) or a REXX command file. The OS/2 Database Manager provides the facility to execute the function on the remote node. This is what differentiates the DARI from other procedure calls that are part of a DLL or command files (see Figure 2).

Figure 3 shows the network data flow for the previous example using DARI. Notice that in order to accomplish the same database manipulations only two network data transmissions are required. The stored function contains the SQL statements and is located at the database server. The data which is used by the stored function is received from the calling application at the database requester. As illustrated, if DARI is used, the network transmissions are dramatically decreased, thereby increasing the total throughput of the LAN.

In addition to reduced network traffic, stored functions can take advantage of the features available on the database server. This includes the Database Manager functions not supported by Remote Data Services, such as the Catalog Node environment API. The user could write a stored function to catalog a node on the remote server. The stored function may also take advantage of the larger memory and the greater disk space typically found on the server machine.

DARI is a very powerful interface allowing the user significant flexibility. User written stored functions can contain code to accomplish any task desired. The functions can range from a "noop" (not involving a database at all) to complicated database manipulations. The power of DARI makes it quite attractive, but one must consider that its flexibility makes its implementation somewhat difficult. This article will provide helpful hints and brings to light the idiosyncrasies and pitfalls that may be encountered when creating stored functions for DARI.

GETTING STARTED

Essentially, DARI is a vehicle which passes data to and from a user generated stored function. A prerequisite to the use of DARI is a connection to a database. The node at which the stored function executes, is where the database physically resides. If the database resides on the workstation, local to the DARI application, then the stored function will be run on that local node.

Programming language support for the DARI function is shown in Figure 4. When using the C programming language, the Database Application Remote Interface is invoked using the following prototype:

```
                /*   DATABASE  APPLICATION  REMOTE INTERFACE */

int far pascal sqleproc(char *,/* Path Name of program to run     */
            struct sqlchar *,  /* Address of variable data area   */
            struct sqlda *,    /* Address of input sqlda          */
            struct sqlda *,    /* Address of output sqlda         */
            struct sqlca * );  /* Address of sqlca                */
```

Table 1

CREATING THE CALLING PROGRAM

When using DARI, the calling program is responsible for the allocation and the initialization of the data areas passed to the stored function. Data is transferred to and from the stored functions using the sqlchar, sqlda and sqlca data structures. All data areas must be allocated by the calling application. Enough memory must be allocated by the calling program to accommodate the expected results from the stored function.

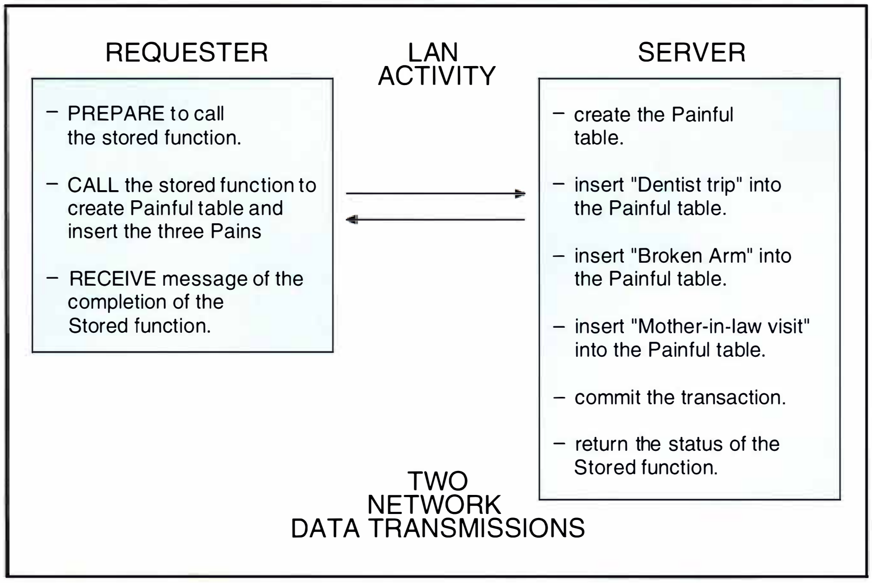

REQUESTER
– PREPARE to call the stored function.
– CALL the stored function to create Painful table and insert the three Pains
– RECEIVE message of the completion of the Stored function.
LAN ACTIVITY
SERVER
– create the Painful table.
– insert "Dentist trip" into the Painful table.
– insert "Broken Arm" into the Painful table.
– insert "Mother-in-law visit" into the Painful table.
– commit the transaction.
– return the status of the Stored function.
TWO NETWORK DATA TRANSMISSIONS

Figure 3. Network Data Transmissions Using DARI.

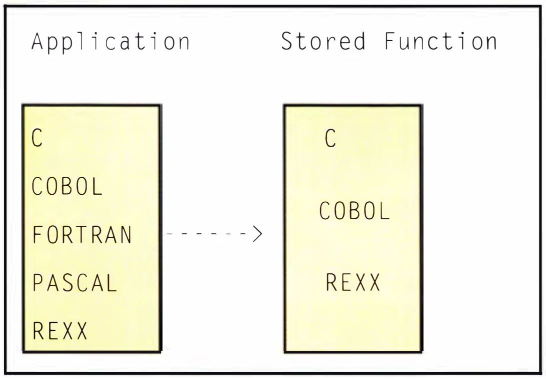


Figure 4. Application Language Support for DARI

The sqlchar, sqlda (SQL Descriptor Area), and sqlca (SQL Communications Area) data structures are found respectively in sqlutil.h, sqlda.h and the sqlca.h files. Their definitions are presented in Figure 5.

The sqlchar data area is used to transmit data to the stored function. This data area is NOT code page translated. There are two sqlda structures on the interface. One is used to pass data TO the stored function (known as the *input sqlda*), and the other is used to pass data FROM the stored function (known as the *output sqlda*). The sqlca data area should be used to pass error data from the stored function although it is not a requirement. The sqlca can also be used to pass information to and from the stored function but it is not recommended. Both sqldas and the sqlca are code page translated by the Database Manager.

Additionally, these data areas are treated differently in the local and remote cases. For performance reasons certain fields within these data structures are not transmitted to a remote procedure. In the local situation the data areas are passed in as parameters to the stored function.

Figure 6 illustrates the data which flows for each data structure in the LAN environment. The left side shows the elements of the data structures which are transmitted from the application program to the stored function. The right side shows the elements of the data structures which are received from the stored function by the application program.

Typically both the calling program and the stored function are developed on a server containing the database. This local/remote inconsistency sometimes results in some confusion while developing the code. The programmer must be aware of the data area differences. To illustrate, consider the sqlname substructure of the sqlda. If the developer sets the sqlname.data field to *OS/2 EE* and the stored function is executed local to the application, the data is passed. If the stored function is on a remote node, the data is not passed. This could result in a segmentation violation of the stored function on the remote node. This is always a debugging nightmare, if the programmer is not cognizant of the facts.

For specific programming examples please refer to the *Avoid the Network Traffic Jam: Passing Blocks of Data Using the DARI* article in this issue of the *IBM Personal Systems Developer*. Other examples can be found in the *IBM Operating Systems/2 Database Manager Programming Guide and Reference, Volume 2: Reference.*

```
struct sqlchar {
  short           length;          /* length of the variable data area */
  unsigned char data[1];           /* pointer to the variable data area*/
};


struct sqlda
{
  unsigned char sqldaid[8];        /* Eye catcher = 'SQLDA   '         */
  long          sqldabc;           /* SQLDA size in bytes = 16+44*SQLN */
  short         sqln;              /* Number of SQLVAR elements        */
  short         sqld;              /* # of used SQLVAR elements        */
  struct sqlvar
  {
    short           sqltype;       /* Variable data type               */
    short           sqllen;        /* Variable data length             */
    unsigned char *sqldata;        /* Pointer to variable data value   */
    short          *sqlind;        /* Pointer to NULL indicator        */
    struct sqlname                 /* Variable Name                    */
    {
      short           length;      /* Name length [1..30]              */
      unsigned char data[30];      /* Variable or column name          */
    } sqlname;
  }sqlvar[1];
};


struct sqlca {
  unsigned char    sqlcaid[8];     /* Eyecatcher = 'SQLCA   '          */
  long             sqlcabc;        /* SQLCA size in bytes = 136        */
  long             sqlcode;        /* SQL return code                  */
  short            sqlerrml;       /* Length for SQLERRMC              */
  unsigned char    sqlerrmc[70];   /* Error message tokens             */
  unsigned char    sqlerrp[8];     /* Diagnostic information           */
  long             sqlerrd[6];     /* Diagnostic information           */
  unsigned char    sqlwarn[11];    /* Warning flags                    */
  unsigned char    sqlstate[5];    /* State corresponding to SQLCODE   */
};
```

Figure 5. Data Structures Used in DARI

CREATING THE STORED FUNCTION

The order in which the data areas are passed to the stored function is illustrated by the following example:

```
int far pascal server(
       struct sqlchar       *vardata,
       struct sqlda         *inda,
       struct sqlda         *outda,
       struct sqlca         *sqlca );
```

Each of these data areas can be NULL. If the stored function contains SQL statements, it must be precompiled and bound to the database that is involved. For more detailed information about precompilation and binding please refer to the *IBM OS/2 EE Database Manager Programming Guide and Reference, Volume 2: Reference.*

After the program is precompiled successfully it is then necessary to compile the C file generated by precompilation. In order for the object module to be a part of a DLL the user must compile the module with the -Alfu option (the -Alfw option can also be used). This produces far (32-bit) pointers and addresses for all code items. The option also reserves different stack and data segments. These are necessary for OS/2 to resolve run time link points into the DLL.

The following is an example of such a compilation statement:

```
        CL -Alfu -c -Gs server.c
```

More information about compiling can be found in the *IBM C/2 Fundamentals, Compile, Link, and Run Debug* documentation. Please note that other compiler options can be used in addition to those previously mentioned.

Data passed from application program to the stored function	Data passed from application program to the stored function
1. INPUT STRING - YES	1. INPUT STRING - NO
2. INPUT sqlda:	2. INPUT sqlda:
sqldaid - NO	sqldaid - NO
sqldabc - NO	sqldabc - NO
sqln - NO	sqln - NO
sqld - YES	sqld - NO
sqlvar.sqltype - YES	sqlvar.sqltype - NO
sqlvar.sqllen - YES	sqlvar.sqllen - NO
sqlvar.sqldata - YES	sqlvar.sqldata - NO
sqlvar.sqlind - YES	sqlvar.sqlind - NO
sqlvar.sqlname - NO	sqlvar.sqlname - NO
3. OUTPUT sqlda:	3. OUTPUT sqlda:
sqldaid - NO	sqldaid - NO
sqldabc - NO	sqldabc - NO
sqln - NO	sqln - NO
sqld - YES	sqld - NO
sqlvar.sqltype - YES	sqlvar.sqltype - NO
sqlvar.sqllen - YES	sqlvar.sqllen - NO
sqlvar.sqldata - NO	sqlvar.sqldata - YES
sqlvar.sqlind - NO	sqlvar.sqlind - YES
sqlvar.sqlname - NO	sqlvar.sqlname - NO
4. sqlca - YES	4. sqlca - YES

Figure 6. Elements of the Data Areas Actually Transmitted Over the Network by DARI

The programmer must also be aware that the function is part of a DLL when linking the object module. To accomplish this, a link and definition file similar to the following should be used:

```
Link file:

          server.obj
          server.dll
          server.map
          LLIBCDLL+SQL_DYN+LLIBCE
          server.def
Definition file:

          LIBRARY server INITINSTANCE
          EXPORTS server
```

More details of linking programs using definition files and building DLLs can be found in the *IBM Operating System/2 Programming Guide* documentation. Please note that other libraries can be included in the link file and other options can be included in the definition file.

After the DLL is successfully created, it must be copied to a subdirectory on the node where the stored function is executed. The subdirectory must be referenced in the systems LIBPATH identified in the config.sys file. For further details on the config.sys file and LIBPATH please refer to the *IBM Operating System/2 Command Reference* documentation.

WHAT WENT WRONG?

Debugging the stored functions can sometimes be a very time consuming and frustrating process. The most prevalent example is the segmentation violation of the server program. Where does one start? The stored functions are executed in background processes. Consequently, there is no way to use Codeview® on the server side to symbolically debug the code of the stored function. The 'printf' statement cannot be used within the code for the same reason. At times it is difficult to tell if the stored function has even been loaded. The following may be helpful in the debugging process:

- If the locally executed program was successful and problems occurred when the program was distributed (application part of the code put on the requester, server part of the code applied to the server), insure that the data used in the stored function was transmitted by Remote Data Services. (*See Creating a calling program section* of this article)

- Using the DOSBEEP OS/2 function call at the function entry point can be used to indicate that the function was entered or exited. For more details of the DOSBEEP operating systems command please refer to the *IBM Operating System/2 Technical Reference Programming Reference: Volume 1.*

- 'fprintf's can be placed at strategic places in the stored function code for printing data, showing the execution path of the program, etc. It is important to give the fully qualified path name of the file so you know where it will be created. For further details about 'fprintf' refer to the *IBM C/2 Language Reference.*

CONCLUSION

The OS/2 EE Database Manager Database Application Remote Interface is a very powerful facility. When used appropriately DARI can greatly increase the performance of a distributed database application. The gains of utilizing this feature far outweigh the *pains* in composing programs that use it (no pain, no gain!). DARI should definitely be considered when designing and developing distributed database applications.

REFERENCES

IBM Operating Systems/2 Database Manager Programming Guide and Reference Volume 2: Reference (01F0269).

IBM C/2 Fundamentals, Compile, Link, and Run Debug (C2CLRBK).

IBM Operating System/2 Command Reference (01F0282).

IBM C/2 Language Reference (C2E5311).

IBM Operating System/2 Technical Reference Programming Reference: Volume 1 (TREFP01).
IBM Operating System/2 Programming Guide (TKPUB01).

Dwayne C. Jacobs, *IBM Entry System Division, Austin Programming Center, 11400 Burnet Rd., Austin, TX 78758. Mr. Jacobs is a development programmer for the OS/2 EE Database Manager product. He has been with IBM for three years and has worked on the Remote Data Services, Base Support Utilities, and the SQL Compiler components of the DBM project. He was a developer for the Database Application Remote Interface. Mr. Jacobs received a BA in Computer Science from the University of Texas at Dallas.*

Extended Edition Database Manager

Avoid the Network Traffic Jam: Passing Blocks of Data Using DARI

Nameeta Sappal

Lance Amundsen

by Nameeta Sappal and Lance Amundsen

This article discusses an advanced technique of passing blocks of data using the IBM OS/2 EE Database Application Remote Interface (DARI). For an introduction to using the DARI, see "Database Application Remote Interface Primer: You Too Can Feel the Power!" in this issue of the IBM Personal Systems Developer.

The Database Application Remote Interface is a powerful facility that can be used to increase the performance of a remote Database Manager application. DARI improves performance by drastically reducing network traffic in the following situations:

- **Remote applications that process large amounts of data and require only a subset of the data to be returned to the user.** Typically, a remote Database Manager application of this type requires that a large number of records be sent across the network. Since a DARI application returns only the data that is needed by the requester, the amount of data transmitted across the network is reduced.

- **Remote applications executing SQL statements that can be grouped together without user intervention.** The more SQL statements that are grouped together, the larger the savings in network traffic. A typical remote Database Manager application requires two trips across the network for each SQL statement. A DARI application requires two trips for each *group* of SQL statements, resulting in a savings of the overhead that would be associated with each trip.

In addition to reduced network traffic, the performance of a remote application that passes arrays, large amounts of data, or packages of data across the network can be improved even more by passing the data in blocks, using the SQLDA data structure as a vehicle for transporting the data.

PASSING BLOCKS OF DATA ACROSS THE NETWORK

Either a requester application or a server procedure can pass a block of data across the network using this technique: A pointer to the data block is placed in one SQLVAR element of an SQLDA data structure using a variable character string data type. The data type can be either SQL_TYP_VARCHAR or SQL_TYP_LONG.

The SQL_TYP_VARCHAR and SQL_TYP_LONG data types are represented in the C language as:

```
struct sqlchar
{
        short                   length ;
        unsigned char           data[1];
};
```

The data field of the sqlchar structure can contain 1 to 4000 bytes for SQL_TYP_VARCHAR and 1 to 32700 bytes for SQL_TYP_LONG.

The SQLCHAR structure includes a pointer to the data and the length of the data. Because the length is specified in the host language representation, the SQL_TYP_VARCHAR and SQL_TYP_LONG data types can be used to pass arbitrary lengths of data such as arrays. The pointer to the data can be dereferenced as required.

A sample program is included in this article to illustrate passing a block of data across the network. The program inserts four new employees into the STAFF table of the SAMPLE database, shown in Figure 1. All four employee's names and identification numbers are passed to the server procedure using an array of data structures. The server procedure inserts the data into the table and returns the result of the entire transaction. This is completed with just two trips across the network — all of the data is passed using only one SQLVAR element of the SQLDA.

The Database Manager program, SQLSAMPL.EXE, creates a database called SAMPLE containing two tables, STAFF and ORG. In order for the sample program in this article to access the SAMPLE database, SQLSAMPL must be executed. For a complete description of the SAMPLE database and the commands used to create it, see the *IBM OS/2 Database Manager Programming Guide and Reference, Vol. 2.*

The name of the file containing the requester application program is requestr.c and the name of the file containing the server program is server.sqc. The following command file is used to build the sample program:

```
cc /AL requestr.c;
link
requestr,,,sql_dyn+sql_stat+llibcp;

sqlprep server.sqc sample
cl /c /Alfu /Gs server.c
link /NOD server,server.dll,,
sql_dyn+sql_stat+llibcdll,server.def
```

The following module definition file is used by the linker to build the server procedure:

```
LIBRARY server
EXPORTS ADD_EMPLOYEES
```

The following data structure definitions are used by both the requester application and the server procedure and are placed in a file called *sample.h.*

```
struct
{
   short id;
   char name[10];
}employee;

struct
{
   short len;
   unsigned short number_of_records;
   struct employee employees[35];
}persons;
```

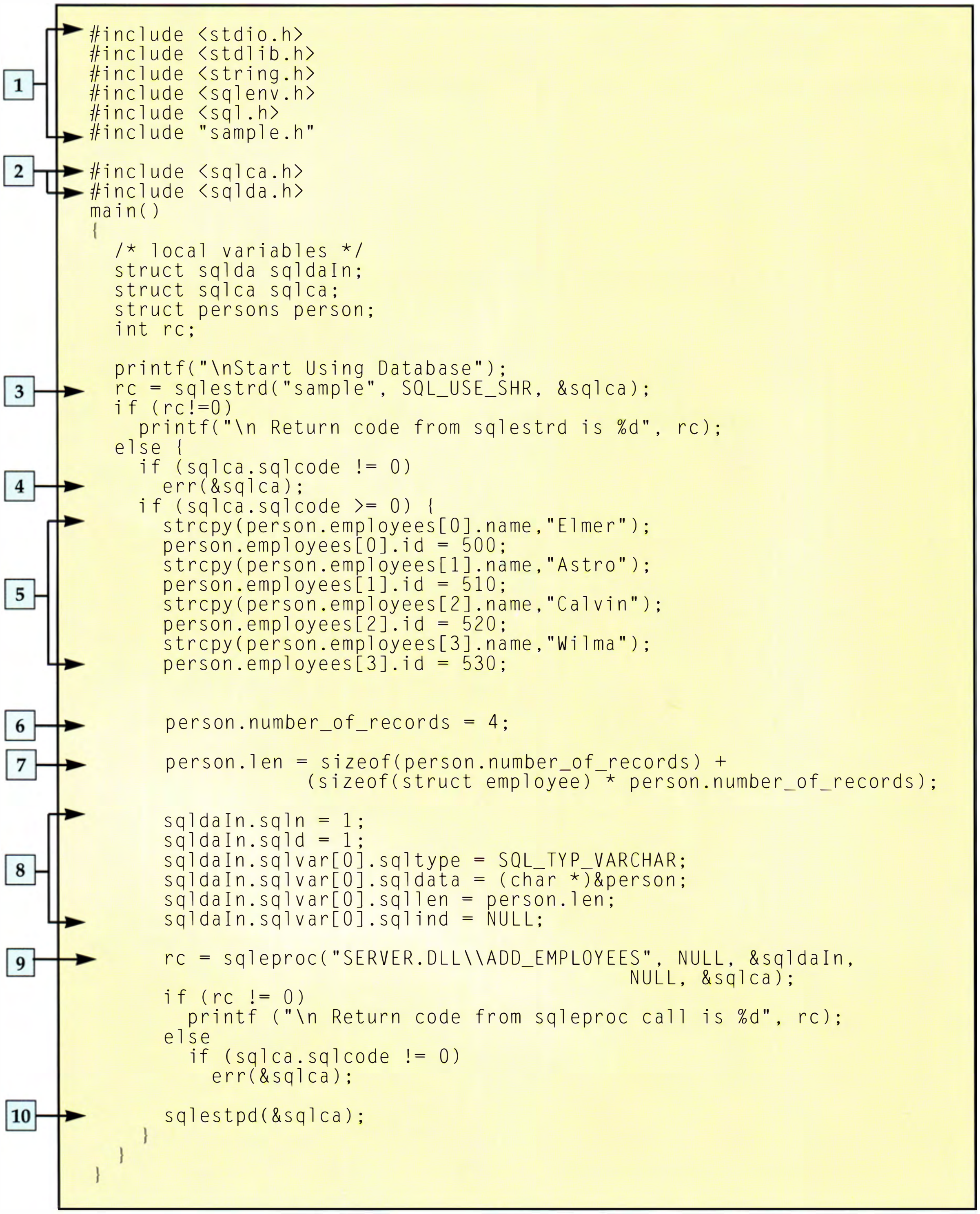
The error handling routine shown below is used by the sample program and calls the GET ERROR MESSAGE *(sqleintp)* routine.

```
void err( ca )
struct sqlca * ca;
{
   char buf[512];
   int rc;

   rc = sqlaintp( buf, 512, 78, ca);

   if ( rc != SQLA_ERR_NOMSG )
      printf(“\n %ld %s”,
      ca->sqlcode,buf);
   else
      printf(“\n Error message for
       sqlcode %ld not found”,
      ca->sqlcode);

   return;
}
```

The Database Application Remote Interface is a powerful facility that can be used to increase the performance of a remote Database Manager application.

ID	Name	DEPT	JOB	YEARS	SALARY	COMM
10	Sanders	20	Mgr	7	18357.50	–
20	Pernal	20	Sales	8	18171.25	612.45
30	Marenghi	38	Mgr	5	17506.75	–
40	O'Brien	38	Sales	6	180006.00	846.55
50	Hanes	15	Mgr	10	20659.80	–
60	Quigley	38	Sales	–	16808.30	650.25
70	Rothman	15	Sales	7	16502.83	1152.00
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.

Figure 1. STAFF Table of the SAMPLE Database


1
2
3
4
5
6
7
8
9
10

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sqlenv.h>
#include <sql.h>
#include "sample.h"

#include <sqlca.h>
#include <sqlda.h>
main()
{
 /* local variables */
 struct sqlda sqldaIn;
 struct sqlca sqlca;
 struct persons person;
 int rc;

 printf("\nStart Using Database");
 rc = sqlestrd("sample", SQL_USE_SHR, &sqlca);
 if (rc!=0)
  printf("\n Return code from sqlestrd is %d", rc);
 else {
  if (sqlca.sqlcode != 0)
   err(&sqlca);
  if (sqlca.sqlcode >= 0) {
   strcpy(person.employees[0].name,"Elmer");
   person.employees[0].id = 500;
   strcpy(person.employees[1].name,"Astro");
   person.employees[1].id = 510;
   strcpy(person.employees[2].name,"Calvin");
   person.employees[2].id = 520;
   strcpy(person.employees[3].name,"Wilma");
   person.employees[3].id = 530;


   person.number_of_records = 4;

   person.len = sizeof(person.number_of_records) +
         (sizeof(struct employee) * person.number_of_records);

   sqldaIn.sqln = 1;
   sqldaIn.sqld = 1;
   sqldaIn.sqlvar[0].sqltype = SQL_TYP_VARCHAR;
   sqldaIn.sqlvar[0].sqldata = (char *)&person;
   sqldaIn.sqlvar[0].sqllen = person.len;
   sqldaIn.sqlvar[0].sqlind = NULL;

   rc = sqleproc("SERVER.DLL\\ADD_EMPLOYEES", NULL, &sqldaIn,
                      NULL, &sqlca);
   if (rc != 0)
    printf ("\n Return code from sqleproc call is %d", rc);
   else
    if (sqlca.sqlcode != 0)
     err(&sqlca);

   sqlestpd(&sqlca);
  }
 }
}
```

Figure 2. Code for Requester Application Program — requestr.c

NOTES ON REQUESTR.C

It is helpful to note the following about the *requestr.c* code listed in Figure 2.

1. **INCLUDE FILES:** The program includes the following Database Manager files:

   sql.h Defines the symbolic SQL_TYP_VARCHAR

   sqlenv.h Contains the prototypes for *sqleproc, sqlestrd, sqlestpd*

   sample.h Defines the data structure *persons.*

2. **INCLUDE SQLDA AND SQLCA:** These statements are used to define the SQLDA and SQLCA data structures.

3. **START USING DATABASE:** The application calls the START USING DATABASE routine requesting shared access. This must be done before invoking the server procedure.

4. **CALL ERROR ROUTINE:** The routine examines an SQLCA data structure for errors.

5. **INITIALIZE DATA:** Each new employee's name and ID number is stored in the data structure *persons.* A pointer to this data structure will be passed to the server procedure.

6. **INITIALIZE THE NUMBER OF ITEMS:** The number of items to be inserted is saved in the data structure *persons* for the server procedure.

7. **INITIALIZE THE SIZE OF ACTUAL DATA:** The first two bytes of the data structure *persons* is the len field which must be initialized to the size of the data following that field, not including the size of the len field itself.

8. **INITIALIZE THE INPUT SQLDA:** The following fields of the input SQLDA are initialized.

   - The *sqln* and *sqld* fields are set to the total number of SQLVAR elements used. NOTE: The SQLDA data structure declared previously allocates space for only one SQLVAR element.

   - The sqltype field is set to SQL_TYP_VARCHAR. If the size of the data exceeds 4000 bytes, then the SQL_TYP_LONG data type must be used.

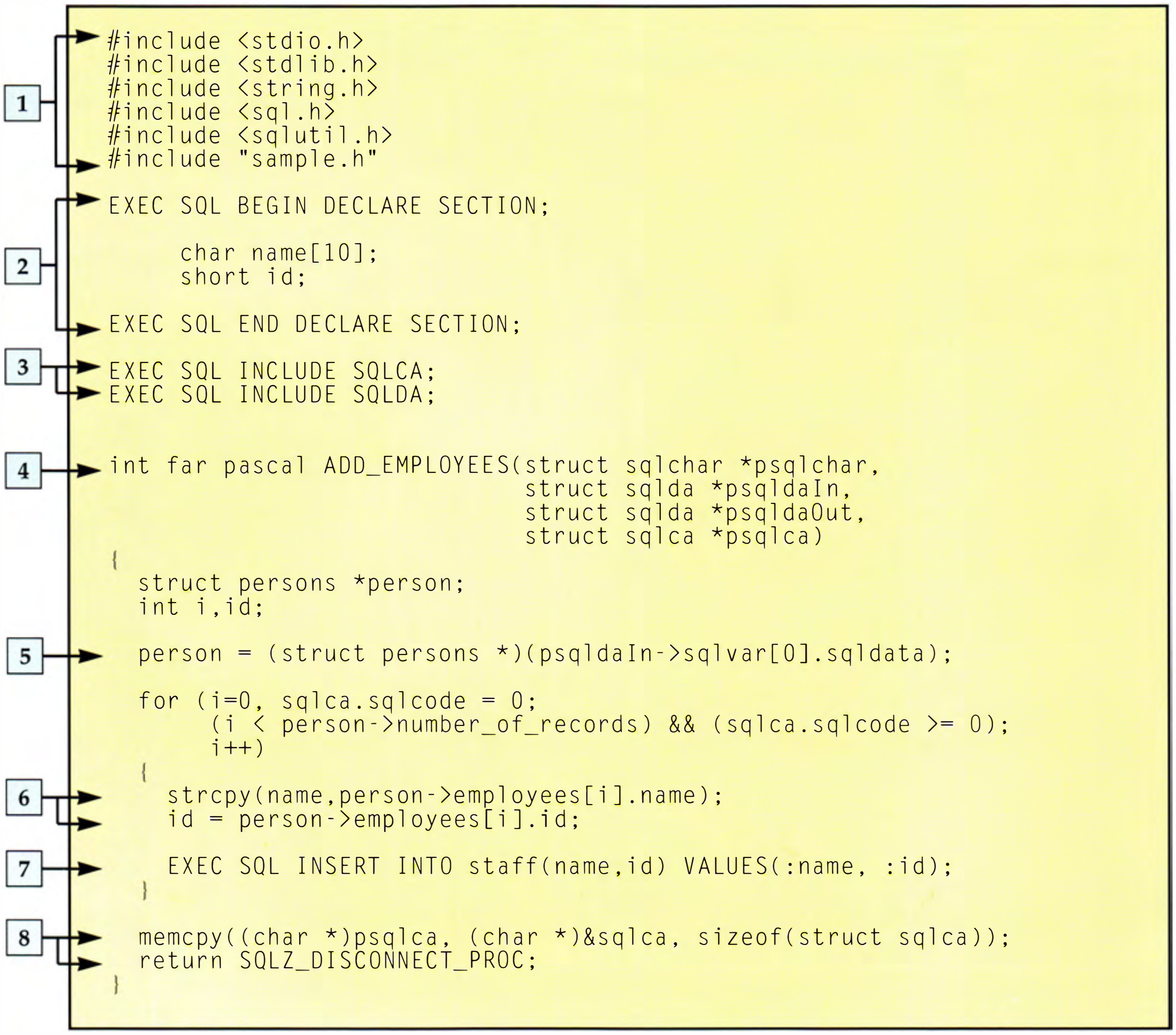
   - The *sqldata* field is set to contain a pointer to the data structure persons. This data structure contains the information needed by the server procedure to perform the INSERT statements.

   - The *sqllen* field is set to the length of the actual data stored in the data structure *persons* minus the two byte length field.

   - The *sqlind* field is set to NULL.

9. **CALL THE SERVER PROCEDURE:** The application invokes the procedure ADD_EMPLOYEES in SERVER.DLL stored at the location of the database SAMPLE. The output SQLDA and the SQLCHAR data structures are not used in this sample. The value NULL is passed in their places.

10. **STOP USING DATABASE:** The application calls the STOP USING DATABASE routine after it is finished processing.



```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sql.h>
#include <sqlutil.h>
#include "sample.h"

EXEC SQL BEGIN DECLARE SECTION;

     char name[10];
     short id;

EXEC SQL END DECLARE SECTION;

EXEC SQL INCLUDE SQLCA;
EXEC SQL INCLUDE SQLDA;


int far pascal ADD_EMPLOYEES(struct sqlchar *psqlchar,
                             struct sqlda *psqldaIn,
                             struct sqlda *psqldaOut,
                             struct sqlca *psqlca)
{
  struct persons *person;
  int i,id;

  person = (struct persons *)(psqldaIn->sqlvar[0].sqldata);

  for (i=0, sqlca.sqlcode = 0;
       (i < person->number_of_records) && (sqlca.sqlcode >= 0);
       i++)
  {
    strcpy(name,person->employees[i].name);
    id = person->employees[i].id;

    EXEC SQL INSERT INTO staff(name,id) VALUES(:name, :id);
  }

  memcpy((char *)psqlca, (char *)&sqlca, sizeof(struct sqlca));
  return SQLZ_DISCONNECT_PROC;
}
```

Figure 3. Code for Server Procedure — server.sqc

NOTES ON SERVER.SQC

It is helpful to note the following about the *server.sqc* code listed in Figure 3.

1. **INCLUDE FILES:** The program includes the following Database Manager files:

   sqlutil.h Define the SQLCHAR data structure

   sql.h Defines the symbolic SQLZ_DISCONNECT_PROC

   sample.h Defines the data structure *persons*

2. **DECLARE HOST VARIABLES:** The host variables are declared in the SQL declare section. These variables are then used to construct dynamically executed SQL statements that use the data passed via the input SQLDA.

3. **INCLUDE SQLDA AND SQLCA:** These statements are used to define and include the SQLDA and SQLCA data structures.

4. **DECLARE SERVER PROCEDURE:** The server procedure, ADD_EMPLOYEES, is declared using the Pascal calling convention. The procedure accepts pointers to SQLCHAR, SQLDA, and SQLCA data structures. The output SQLDA and the SQLCHAR data structures are not used in this case.

5. **REFERENCE THE INPUT DATA:** Cast the pointer in the first SQLVAR element of the input SQLDA as a pointer to the data structure persons and initialize the local pointer person to point to that data. The data can then be accessed using this pointer to the data structure.

6. **INITIALIZE HOST VARIABLES:** Copy the person's name and ID number into the corresponding host variables, *name* and *id.*

7. **INSERT DATA:** Insert the names and ID numbers into the table STAFF.

8. **RETURN TO THE REQUESTER APPLICATION:** The server procedure copies the SQLCA to the SQLCA of the requester application. It then returns the value SQLZ_DISCONNECT_PROC which indicates that no further calls to the server procedure will be made.

CONCLUSION

Passing blocks of data using DARI is most powerful when used in host languages supporting data structures. A data structure or an array of data structures representing complex data can be passed across the network, as illustrated by the sample program. Large amounts of data can also be passed in this manner.

The technique of passing blocks of data can be exploited beyond the sample program described in this article. It allows an application to customize record blocking that is optimized for a particular application environment. This can greatly improve the performance of a remote database application.

REFERENCES

IBM. *OS/2 Database Manager Programming Guide and Reference.* (S0IF-0269). Available through Branch Offices.

"Database Application Remote Interface Primer: You Too Can Feel the Power", IBM *Personal Systems Developer, Spring 1991.*

Nameeta Sappal, *IBM Personal Systems Programming Center, 11400 Burnet Road, Austin, TX 78758. Ms. Sappal is an Associate Programmer for the OS/2 EE Database Manager Product. She has been with IBM since May 1989 and has worked primarily as a developer for the SQL Compiler team. She received a BA in Mathematics/Computer Science from the University of California at San Diego.*

Lance C. Amundsen, *IBM Personal Systems Line of Business, 11400 Burnet Road Austin TX 78758. Mr. Amundsen is a Senior Associate Information Developer. He joined IBM in 1989 as an Information Developer for the OS/2 EE Database Manager project and is the current author of the OS/2 EE Database Manager Programming Guide and Reference. He received a BS in Physics from the University of Minnesota in 1985.*

Extended Edition Database Manager

Client-Server Databases: Three Approaches

Charles McKelley

by Charles McKelley

Version 1.2 of the IBM Operating System/2 Extended Edition Database Manager introduced two features, Remote Data Services and Database Application Remote Interface, which make it easier for the programmer to build client-server applications on top of the OS/2 EE DBM program. Prior to version 1.2 the only way to build client-server applications on top of the database manager was through user-written distributed applications. These are sometimes referred to as 'roll your own' solutions. This article discusses the tradeoffs to be considered when deciding which approach to use for client-server applications based on the OS/2 EE DBM program.

Before we start it is necessary to define a couple of terms that will be used throughout this discussion.

A *client workstation* is a PC which is used to run application programs. It contains a requestor program, which sends requests over a local area network (LAN) to the server workstation.

A *server* is a workstation containing the database kernel and its associated databases. The catcher program resides on the server and receives requests from the requestor.

Dynamic SQL is parsed and processed by the database manager each time the application program is executed.

Static SQL is imbedded in the application source code. It is parsed and processed by the database manager precompiler prior to execution time so it is normally faster than dynamic SQL.

A *unit of work* is a set of database operations that makes up a single transaction. This might not seem to fit in with the definitions above, but it takes on special significance in a client-server context. If the connection between the client and server is interrupted when part, but not all, of a unit of work has been applied to a database, the client-server programs must be able to detect that condition and to recover from it in order to maintain the integrity of the data.

Unit of work recovery is one important objective of client-server database programs. Here are some others:

- Application program and end-user access to the remote data should be controlled by some authorization mechanism.
- The location of the database should be transparent to the application program.
- The application program should be isolated from the transport protocol so that, if the transport protocol is changed, or if multiple transport protocols are used, the application program need not be rewritten or contain logic specific to the transport protocol.
- Traffic on the network should be kept to a minimum. There are two aspects to this problem:

First, the number of requests that flow across the network should be minimized.

Second, the size of the data packets that flow on the network should be kept as small as possible.

Now let's look at each of the three approaches available to accomplish these objectives.

ROLL YOUR OWN

Roll your own (RYO) solutions to the problem of distributing SQL requests generally fall into one of two categories.

The first, and most difficult, method involves intercepting the calls to the database manager at the Application Program Interface (API) level, marshalling the parameters, and shipping them to a catcher on the server workstation. This method includes some difficult and complex code and is beyond the scope of this article.

Editor's note: Dana Beatty's article on the NetBIOS API appears in this issue.

The easiest way to roll your own client-server solution is to provide a requestor which can package an SQL statement or database environment request as a character string and ship it to a catcher on the database server workstation. The catcher submits the request to the database manager SQL API using dynamic SQL. In this design the catcher is really a database application program. The client application is *not* really a SQL application program. It is *not* processed by the database precompiler nor is it bound to the database. Therefore, it is invisible to the database manager.

See Figure 1 for an illustration of the model for this RYO design.

The major steps which must be accomplished are:

A. The requestor must determine where the desired database and server are physically located.

B. The client application program must communicate to the catcher its desire to retrieve data from a database on the server. This may be done explicitly via a connection request or it may be implied by a data retrieval request.

C. Either the requestor or the catcher must verify that the application program and the end-user have authority to access the database.

D. The requestor must transfer the request for data from the application program to the catcher. The catcher must submit the request to the database manager SQL API at the server workstation and return the answer to the requestor, which in turn must pass it back to the client application program.

The box below illustrates the commands the catcher must issue if the request in step D is a SELECT resulting in multiple result rows.

```
DECLARE CURSOR
PREPARE SELECT
OPEN CURSOR
FETCH 1
         .
         .
         .
FETCH n
CLOSE CURSOR
```

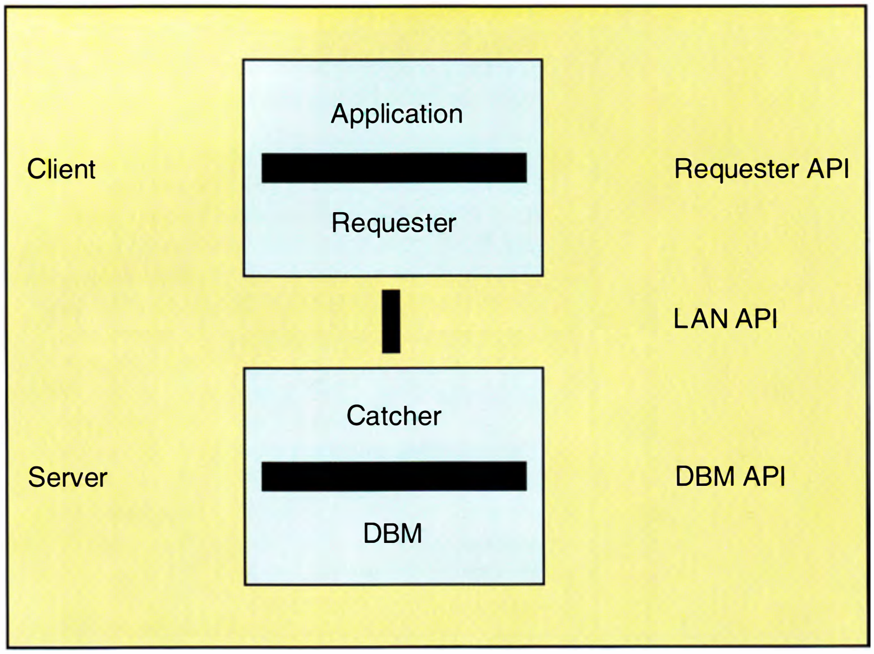


Figure 1. RYO System Structure

Notice that, since the catcher must use dynamic rather than static SQL to communicate with the database manager, it must issue the PREPARE statement to allow the database manager to parse the SELECT statement before it is executed. This can result in an extra call to the database kernel for every SELECT the application program wishes to execute. It also means that syntax errors are detected at run time rather than at precompile time. Notice also that the catcher receives multiple rows from the database kernel in response to the FETCH statements it executes. If all of the rows are returned to the requestor in a single block, some means must be used to ensure that the requestor buffer does not overflow. In addition, if the rows are returned to the application program in a complex data structure, the application must understand the format and must contain logic to parse the row data.

The catcher must manage the serialization or multiplexing of requests from multiple application programs. The database manager will handle the multiplexing of concurrent instances of the catcher program running in different server processes, but the catcher must keep track of which instance is paired with a particular application program request. This situation can become complex if multiple applications are permitted to run concurrently on the client workstation.

Authorization must also be handled by the catcher in this design. It is the catcher's responsibility to ensure that the user and application program are authorized to access the server and database. In addition, since the catcher is the only program which is logged on to the database manager, all transactions occur under the userid with which the catcher has logged on. This does not allow the catcher or application program to take advantage of the database manager's GRANT/REVOKE authorization mechanisms. This can be rectified through the use of a multi-process catcher, although that adds to the complexity of the catcher code.

The burden for unit of work recovery is also on the client-server code with this approach. It must be cognizant of the transaction state of each application program instance so that, if the link to a client workstation should fail, the current transaction for that application can be aborted.

Despite the fact that a great deal of code must be written and many complex problems must be solved, there are some advantages to using the RYO approach.

The OS/2 EE DBM program client-server features discussed later in this article support a finite set of transport protocols. The RYO approach allows the support of different or additional protocols. It may also allow the addition of third-party network enhancements such as name servers, etc.

REMOTE DATA SERVICES

Remote Data Services (RDS) is the OS/2 EE DBM program's integral client-server support. It is implemented beneath the SQL API. For this reason applications designed to use RDS are written just as though they were intended to run on a stand-alone database manager. There are no special commands used to direct operations to a remote server. Location transparency is accomplished through the use of directories, which are external to the application program. See Figure 2 for an illustration of the RDS system model.

RDS addresses all of the client-server objectives listed above. Authorization is controlled by the User Profile Management (UPM) function and the underlying communication protocol. When a user executes a database application program against a remote database, the database manager requestor forces the user to logon to the system through UPM. Once the logon is performed at the client workstation, RDS interacts with the underlying communication functions to verify conversation authorization and to log the client on at the server workstation. If the userid at the client workstation is not authorized at the server, RDS will deny access.

In addition, the application program is processed by the database precompiler. This allows the application program to be bound to the database so that only authorized programs will access the data. This also allows the use of static SQL, which offers a performance advantage over dynamic SQL, since parsing and syntax checking of the statements take place at precompile time.

RDS manages multiple requestors on a LAN as well as multiple instances of the requestor on a single OS/2 workstation. The application program is shielded from the transport protocol and from the data interchange format so that enhancements in future versions of the database manager or changes in the transport protocols will not require the application programs to be changed or recompiled.

RDS performs intelligent transfer of data between the client and server in order to minimize traffic on the network. For example, if a query returns a large result table consisting of many rows, RDS can send the answer set back to the client in blocks. Then as the client application program retrieves each row of data using SQL cursor operations, RDS passes the data to the application from the local cache buffer in the client workstation. Additional requests are sent to the server only when no more complete rows exist in the local data buffer.

The size of the data packets is minimized before transfer by first converting the control blocks and data buffers to a data stream which is an optimized version of Distributed Remote Database Architecture. DRDA is IBM's strategic SAA architecture for the exchange of data between relational database manager requestors and servers.

RDS support is available on two operating system software platforms: OS/2 and DOS. There are some differences between the requestors on the two platforms that the programmer should understand.

The RDS OS/2 requestor program is implemented as dynamic link libraries (DLLs), which are loaded and executed on demand. The RDS DOS requestor is implemented as static link modules which become a part of the physical application program. The DOS requestor modules for a given function are linked into the application program only when that function is called by the application. This minimizes the size of the application program.

The communication support for both the DOS and OS/2 program RDS requestors, as well as the database server, is supplied by OS/2 EE. It is not necessary to use the LAN Server to support the database clients and servers. The OS/2 requestor program supports IBM's Advanced Program to Program Communication protocol (APPC), supplied by the OS/2 EE Communication Manager (CM) and a special protocol optimized for the LAN environment called the Structured Query Language Local Area Network Only Option (SQLLOO). The DOS requestor requires the NetBIOS support provided by the LAN Support Program. The server requires the OS/2 EE CM program APPC or SQLLOO support and, if DOS requestors are used, the CM NetBIOS support.

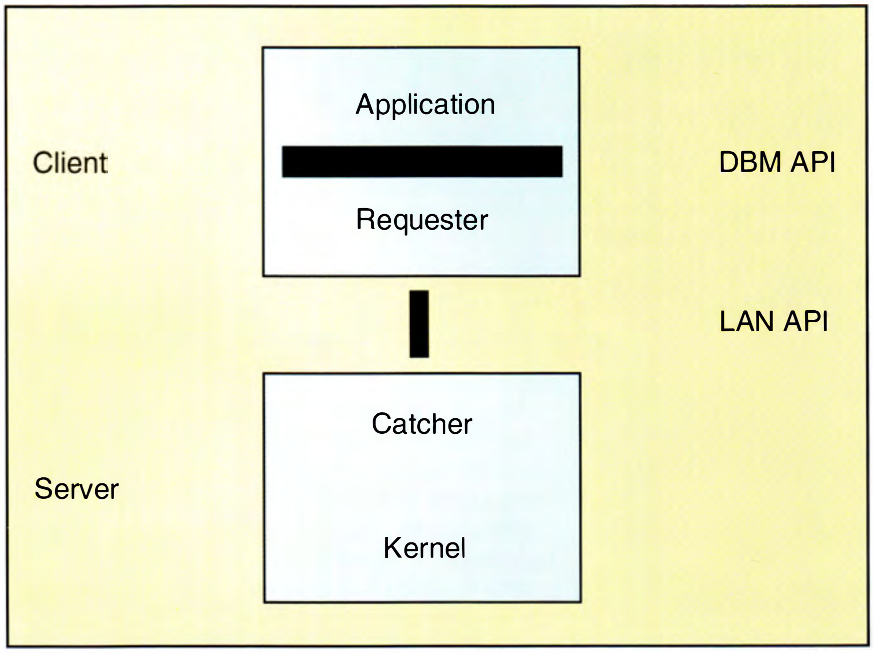


Figure 2. RDS System Structure

Although the transport protocols are invisible to the application program, the use of NetBIOS on the DOS platform is important to the programmer because its use means that much more space is available to the application program than would be available if APPC-PC were used.

It is also important to the programmer to understand that the DOS requestor uses the synchronous call interface to NetBIOS. Use of the NetBIOS interface in this manner requires the DOS requestor to supply NetBIOS with a control block from the application data space. This means that the application program modules containing the DOS requestor code must not be paged out of memory. For this reason the DOS requestor code should be treated like device driver code. That is, in extended memory parlance, it should always be packaged *below the line.* The DOS requestor also contains a terminate and stay resident (TSR) routine which implements the logon/logoff functions. This TSR must also be located below the line.

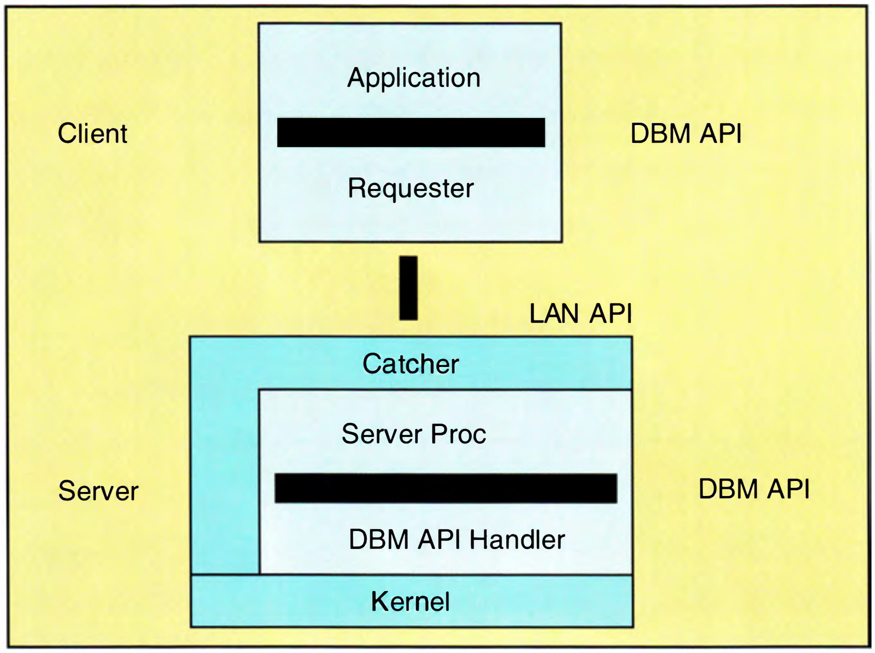


Figure 3. DARI System Structure

Most of the functions available on the OS/2 requestor program are also available on the DOS requestor. One set of functions not available on the DOS requestor is the set returning user status. These functions are intended to be used by database administrators who are not likely to use DOS workstations.

One last note on the RDS DOS requestor: It is important to note that the DOS database requestor was designed to operate only in the DOS environment. It is not intended to work in any other environment.

DATABASE APPLICATION REMOTE INTERFACE

Another new database feature, which can be used to dramatically reduce the traffic on the LAN, is the Database Application Remote Interface (DARI). This powerful feature extends RDS to allow procedures to reside and execute *at a server* on behalf of a remote RDS requestor. The DARI feature is intended to be used to perform often repeated sequences of SQL commands at the server. The result of the entire sequence can be returned to the client in one operation. This eliminates many network calls which significantly reduces the traffic on the LAN.

See Figure 3 for an illustration of the DARI model.

The client application issues a DARI function call to the DBM API in the client workstation. RDS transmits that request to the catcher in the server, which in turn causes the server procedure to be executed. The server procedure issues SQL commands to the DBM API in the server machine. When the catcher procedure completes, RDS returns an SQL Data Area (SQLDA) and any associated data to the client application program.

The client procedure can be written in any language supported by the OS/2 EE DBM program. The server procedure must be written in REXX/2, COBOL/2 or C/2. The server procedures need not be registered or cataloged. They need only exist as DLLs or REXX2 procedures in some directory located by the CONFIG.SYS file LIBPATH variable.

Since the DARI is implemented on top of RDS client-server function it provides many of the same advantages. For example, normal RDS authorization and location transparency mechanisms are in effect for DARI operations. In addition, RDS transport protocol independence is in effect in the DARI environment. Some RDS functions, such as row blocking, are not meaningful for DARI operations.

The DARI interface allows the client application to pass a 32KB buffer and two SQLDA array variables to the server procedure. The server procedure can use an SQLDA to transmit multiple data areas back to the client procedure. This allows for fast and efficient transfer of large blocks of data in both directions. The OS/2 EE DBM program uses this method to import files residing on the client workstation into databases on the server with minimal impact on network traffic.

The DARI feature is extremely powerful. Since the procedures are written in programming languages such as C/2 and REXX/2 rather than a proprietary database language, they are not limited to performing SQL functions. The server procedure can do anything within the capability of a general purpose REXX/2, C/2 or COBOL/2 program. For example, it can be used to integrate data from several sources and to present the data to the client application program in a consistent manner via an SQLDA.

The DARI function provides the advantages of superior performance and smaller application program size. It is also useful for minimizing application program differences between OS/2 and DOS client application programs.

The DARI function works in the same manner whether the client application program operates on a remote machine or on a server. In the case where the application program operates on a server, the DARI function bypasses RDS and delivers the request straight to the database kernel, which then calls the server procedure.

The fact that the DARI function uses RDS for client-server communications means that DARI function calls can be mixed with other database calls in an RDS environment, allowing a single application program to make optimal use of features such as row blocking in addition to the DARI features.

The DARI server procedure has one requirement that does not exist for the client procedure. Since the server procedure can be used concurrently by multiple client procedures, it must be reentrant. That happens automatically if REXX/2 is used for the server procedure. If the COBOL/2 or C/2 programming language is used to build a DLL for the server procedure, care must be taken to ensure that the logic is reentrant and that the proper compile and link options are used to retain the reentrant nature of the DLL.

SUMMARY

Table 1 lists the tradeoffs to consider when choosing a client-server approach for applications based on the OS/2 EE DBM program.

	RYO	RDS	DARI
Server authorization managed by DBM	N	Y	Y
Able to use static SQL	N	Y	Y
Database location transparency provided by DBM	N	Y	Y
Application isolated from transport protocol	?	Y	Y
Application row access via cursors	N	Y	Y[1]
UOW recovery managed by DBM	N	Y	Y
Network traffic minimized	N	Y	Y[2]
Multiple application program instances managed by DBM	N	Y	Y[3]
Require separate communication subsystem	?	N	N
Multiple SQL requests per client call	?	N	Y

?	Dependent upon implementation
1	Only server procedures can use cursors
2	Very low network traffic volume
3	Server procedure must be reentrant

Table 1 Tradeoffs

Of the three approaches discussed here, RYO solutions offer the greatest flexibility but place the greatest burden on the programmer. The skills required to implement this approach are more in the realm of the system programmer rather than the application programmer.

The RDS approach results in the least amount of application code. Applications which use RDS can be written in the same manner as stand-alone database applications. The only additional work is the effort to catalog remote nodes and databases. That level of work is usually done outside the scope of application programs.

The RDS approach is also the most extensible from an SAA specification perspective, since the client-server function is implemented within the database manager and under the SQL API.

Although the DARI approach is not part of SAA, it has some additional advantages over the RDS approach. DARI programs can result in smaller client application programs as well as extremely low network traffic. In addition, they offer the opportunity to use the server as a point of control for the integration of data from various sources, utilizing the inherent DBM authorization functions.

More detailed information on RDS and the DARI can be found in the OS/2 *EE Database Manager Programming Guide and Reference* (S01F0269). Two articles about DARI also appear in this issue.

Charles R. McKelley, Jr., *IBM Austin Programming Center, 11400 Burnet Road, Austin, Texas 78758. Mr. McKelley is an Advisory Programmer on the OS/2 EE Database Manager competitive evaluation team. In his twenty-three years with IBM he has held various positions in field support and software development. He joined the OS/2 EE Database Manager project in 1986 where he has served as client-server architect and Remote Data Services development manager.*

Application Enablers

CICS OS/2 as an Application Enabling Platform

by Harry Starkman

Cooperative Processing Comes to the PS/2!

Imagine, if you will, the following scenarios. You are an independent applications developer working primarily with PS/2 based applications. You have been asked to submit a systems proposal to a very large and influential company. The success of this project could guarantee your future as an independent developer. OR...you are the senior analyst of a mainframe CICS™ development group and have been chosen to work on a project the success of which could lead to a Vice President's title for you.

The project? Simple. The company plans to purchase a large number of PS/2s and they want you to take their largest, most important money-making application and incorporate the PS/2s into it. The application will run at the host and at the PS/2 workstations. Although the main data bases will be at the mainframe, each programmable workstation will have local data. Data must be accessible from either direction. Several functions of this application will be updating files at the mainframe and the workstation, so data integrity and synchronization are critical. Additionally, to preserve network performance you must keep display screens at the PS/2 and send only file data around the network. Most of the PS/2s will be on LANs.

Of course none of this could ever happen in real life...or could it?

The purpose of this article is to introduce CICS and CICS OS/2 and describe some of the major functions they provide for the application developer as an application enabling system. The review of these functions will be at a very high level — a view from 10,000 feet. Detailed information can be found in the various CICS manuals. A starter list is provided at the end of this article.[1]

Harry Starkman

CICS, which stands for Customer Information Control System, has been available for large and mid-range computer users for almost 22 years. One of IBM's most popular software products with more than 30,000 licenses in place worldwide, CICS can run with the IBM MVS, VSE, or VM operating systems and now with OS/2 EE or IBM's PC DOS. In this article, the term CICS, unless otherwise stated, refers generically to both the mainframe and OS/2 versions.

CICS is an Online Transaction Processing (OLTP) System.

ONLINE TRANSACTION PROCESSING

CICS is an Online Transaction Processing (OLTP) System. This distinguishes CICS from batch or background systems. OLTP means that terminal operator requests (transactions & tasks) are performed by CICS in real time. The results of actions against data are in effect at the time of change, not after some overnight processing. For example, a bank teller using a display terminal can view my current account balance and be assured of up-to-the-minute figures, rather than relying on bulky printouts of yesterday's figures.

I can hear the PS/2 application developer saying, "but that's how all PC applications work — in real time." This is true, but only for the single user. Although to its operating system CICS appears as a single image, that

[1] For a full history of *CICS* see *Customer Information Control System- An evolving system facility* by B.M. Yelavich in *IBM SYSTEMS JOURNAL Vol. 24, NOS 3/4, 1985*

is, as one job or program, under the internal control of CICS, hundreds of tasks (units of work) per second are being done in real time. A typical CICS mainframe system can have thousands of display devices, with as many users, attached to a single copy of CICS —all working in real time! A single PS/2, a model 80 for example, can have up to three ASCII terminals attached to it, each one defined to the CICS OS/2 system. The end users at the ASCII terminals are now able to execute CICS transactions, performing multi-user processing on a single CICS OS/2 system.[2]

API

For the program developer the heart of CICS is the command level Application Programming Interface (API). The API commands are CICS program statements such as the file READ function in Figure 1. The following example shows, in its simplest form, the command used to read a record from a file named MASTER into a data area named RECORD:

```
EXEC CICS READ
       INTO(RECORD)
       FILE('MASTER')
       RIDFLD(ACCTNO)
```

RECORD and ACCTNO are data description items. The file name MASTER would be defined in a CICS Control Table[3]; here it would be the file control table, where the file's fully qualified data set, name is defined. In this way a resource is only defined once and can be used by many transactions. Also, the RIDFLD, which stands for record identifier field, represents a means of selecting a record with a specific identifier key. Here's the best part: the API for mainframe CICS and the API for CICS OS/2 are the same in use and form. The READ command looks and works the same on either the host or the PS/2. With CICS running on both systems, cooperative processing becomes an easily attained goal. Let's now look at some of these CICS functions.

TERMINAL COMMUNICATIONS

You sit down at a computer keyboard and start a program. A message from the program appears on the video display and you type a reply. The process of carrying on this conversation is referred to as data communications. Programs that use an exchange of data from the keyboard/screen require that the programmer know much about the terminal's characteristics as well as telecommunications line protocols. As a company's network changes programmers must modify their programs.

Imagine writing a program for one type of terminal and after the program is complete, being told that several new devices were just added to the network. The program must now be changed to determine which device it is dealing with. This adds to program complexity and raises the possibility of program errors.

The CICS facility that aids a program with data communications is Terminal Control. Terminal Control has two functions that we are concerned with (T.C. has other functions but they are outside the scope of this article). The first is to define the display devices that are attached to the system. This is done through a terminal control table. Each terminal has a terminal control table entry that defines its characteristics to CICS. Now the various CICS programs can communicate with the terminals without dealing with terminal differences themselves; this gives CICS applications device independence.

The second feature of Terminal Control that we will discuss is the actual sending and receiving of terminal data. Here too, CICS gives us two options. The first is using terminal control commands to perform terminal reads or writes of some data. This data can be a single line or a full screen of data. However, using Terminal Control commands requires that the application programmer be aware of data formatting both in how the message is sent to the screen

[2] *There are limitations on the type of CICS OS/2* applications *that can be used with an ASCII terminal.* See the *CICS OS/2 Version 1.20 Systems* and *Applications Guide for full details.*

[3] *CICS* is a table driven system. All the files, terminals, programs, transactions, queues, and other resources that are used by *CICS* applications are defined through *CICS* control tables. We will refer to the specific table within each function.

and how it is displayed on the screen. This means that if a programmer wanted to use features such as highlighting, underscoring, or reverse video the program would need to insert the control characters into the data stream before the data could be written.

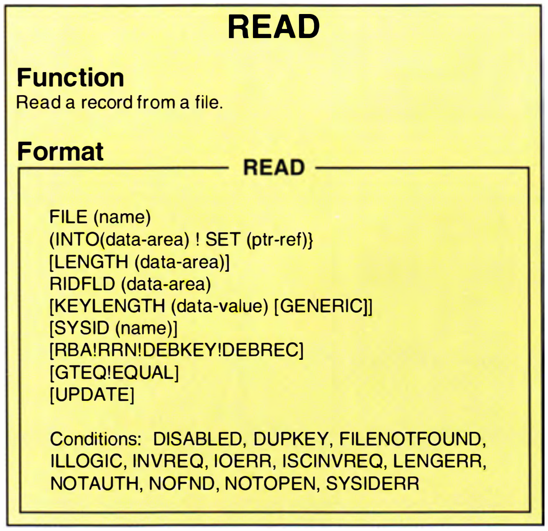
Our second data display option is Basic Mapping Support (BMS). BMS controls the way data is displayed at the terminal and how the program identifies it. It is an interface between CICS application programs and terminal control functions to provide both device and format independence. It lets the programmer separate the tasks of display design and CICS application programming. BMS interprets generalized device-independent application program output commands and generates device-dependent data streams for specific output devices. It also transforms incoming data streams to a form acceptable to application programs. It learns about the format of the data stream for the terminal from the terminal control table terminal entry, not from the application program.

BMS provides these functions through the use of MAPS. Each map has two forms, physical and symbolic. The physical map tells BMS how to format the display and the symbolic map helps the application program reference the fields in the display. The maps are easily built using CICS supplied macro instructions. Both BMS commands and terminal control commands can be used in the same program making for greater terminal display flexibility.

With CICS OS/2 the application developer can use either of these methods of screen display. CICS OS/2 also lets the developer use the OS/2 Presentation Manager directly. Like BMS, Presentation Manager allows screen design to be independent of the application program. Calls to PM can be made from a CICS OS/2 program. Unlike BMS, PM is not physically a part of CICS, it is a service component of OS/2 that can be called on by CICS applications. When BMS or Terminal Control is used for terminal display, a CICS application program can execute on either a PS/2 or a mainframe. When PM is used, the program can be used on the PS/2 only. However, using PM gives the program a familiar look and feel to the user, and makes it compatible with Systems Application Architecture (SAA).

DATA HANDLING

Data manipulation is the purpose of most application programs. In the batch realm, data handling can be a very tedious task. The programmer must be well versed in the control and data structures of the various data access methods. At the mainframe level, CICS applications have access to standard files such as Virtual Storage Access Method (VSAM). CICS File Control gives the application developer VSAM access without having to worry about a data set's type or physical organization. A file's characteristics are defined to CICS in a File Control Table entry. A CICS application program reads and writes its data in the form of individual records (see Figure 1 for the READ command).



READ

Function

Read a record from a file.

Format

READ

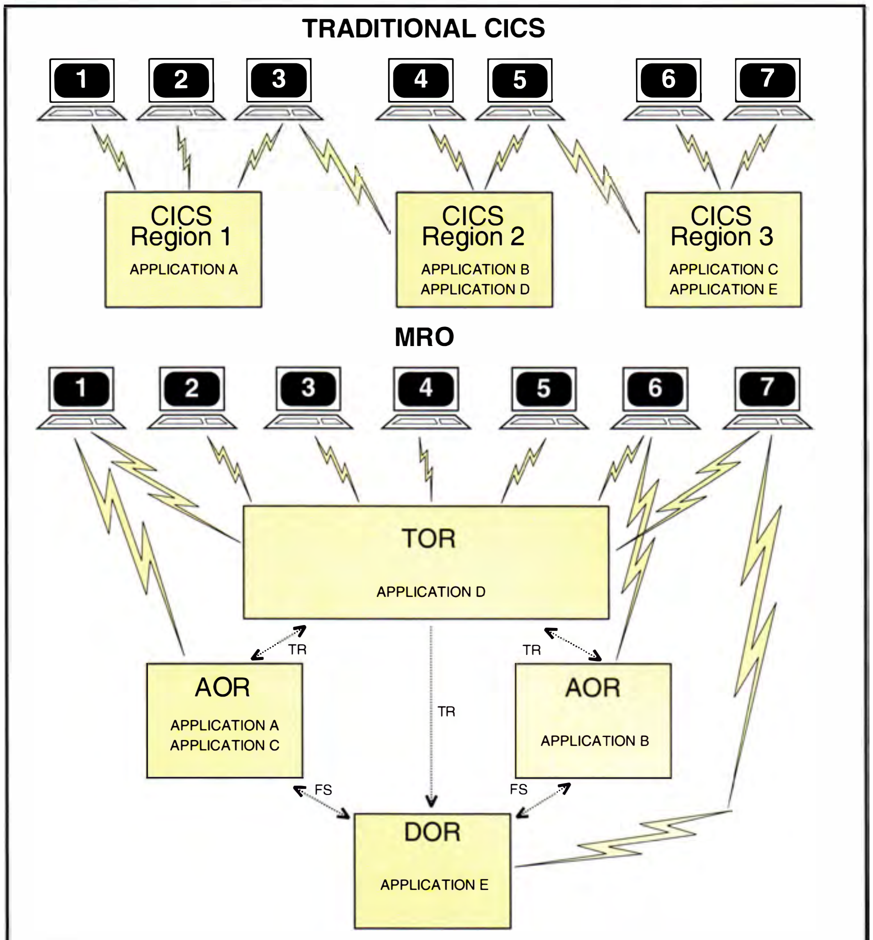
```
FILE (name)
(INTO(data-area) ! SET (ptr-ref)}
[LENGTH (data-area)]
RIDFLD (data-area)
[KEYLENGTH (data-value) [GENERIC]]
[SYSID (name)]
[RBA!RRN!DEBKEY!DEBREC]
[GTEQ!EQUAL]
[UPDATE]
```

Conditions: DISABLED, DUPKEY, FILENOTFOUND, ILLOGIC, INVREQ, IOERR, ISCINVREQ, LENGERR, NOTAUTH, NOFND, NOTOPEN, SYSIDERR

Figure 1. CICS READ Command

CICS OS/2 provides the application developer with VSAM-like facilities through an internal file manager, which makes for easy program portability between mainframe and PS/2. The developer should note, however, that such files are not accessible outside of the CICS OS/2 environment. CICS provides file integrity through transaction backout and also provides, outside of File Control, anenqueuing facility that gives the application developer even greater control over file access. This way, if you must update several files, you can insure that no other file access is permitted from other transactions until you have completed all your work.

At the host we can also use hierarchical and relational database systems. These two types of databases are represented by IBM's Data Language/I (DL/I), which is implemented by IMS/VS and by Database 2 (DB2) and Structured Query Language/Data System (SQL/DS), respectively. Mainframe CICS provides interfaces for DL/I and DB2 database access. A single mainframe CICS application program can use one or all of these file and database methods.



TRADITIONAL CICS

1. TERMINALS 1 and 2 can only access Application A in Region 1. TERMINAL 3 can access Application A in Region 1 and Applications B and D in Region 2. To do so the terminal must logoff Region 1 and logon to Region 2.
2. Same for Terminal 5 between Regions 2 and 3.

MRO

3. All terminals are attached to TOR. They can access Application D in the TOR or use transaction routing to access the other applications. Also Terminal 1 can logoff TOR and logon AOR. Terminal 7 can do the same with the DOR.

Figure 2. Traditional and MRO CICS Systems

At the PS/2 the developer can code SQL statements in a CICS OS/2 program to access local or remote databases through OS/2 Extended Edition's Database Manager. A single CICS OS/2 application program can access both VSAM filesand SQL databases. When we add in intercommunication facilities a mainframe CICS application can access PS/2 data through CICS OS/2 just as a CICS OS/2 application can access the mainframe.

DATA INTEGRITY

How do you protect your data from accidental loss or corruption? What can you do about maintaining the integrity of your data during updating, if the program or the operating system fails? Data integrity is made up of several functions. One is to ensure that a record is updated by only one user at a time. CICS ensures that one operator's updating is complete before another's may start.

During normal execution of a task, CICS keeps information about all protected data that is being changed. Protected data is called a recoverable resource. If for any reason a transaction can't complete and it has made changes to a protected file, the data changes can be reversed, restoring the protected data to its original state. If a record was added, it is removed; if deleted, it is returned; if updated, it is restored.

Dynamic transaction backout (DTB) is the CICS facility that manages this recovery. The process of canceling changes in data works backwards from the last change before the failure. The backout occurs within the same task. This safeguards other tasks from the possibility of using corrupted data, because CICS does not release modified data for use by other tasks until the current task ends or the change is committed.

However, there can be times when a change to a protected resource must be committed before the program ends. To do this CICS provides the SYNCPOINT command. This will commit any outstanding changes and these changes will not be backed-out if the program fails before ending. Dynamic transaction backout and syncpointing provides invaluable support when your application is doing multiple data updating across systems.

For example, let's say your application must update two files before it ends. If after making the changes to both files, but before these changes are committed, the program or system fails, DTB will back out the changes on both files. No special coding is required if the resources were defined as protected. Also, if you want the changes committed before the applications end then issuing the SYNCPOINT command will commit the changes.

If the entire system were to fail — power outage is a good example — CICS provides the means to do an emergency restart, one function of this being DTB. All uncommitted (the CICS phrase is in-flight) changes will be backed-out.

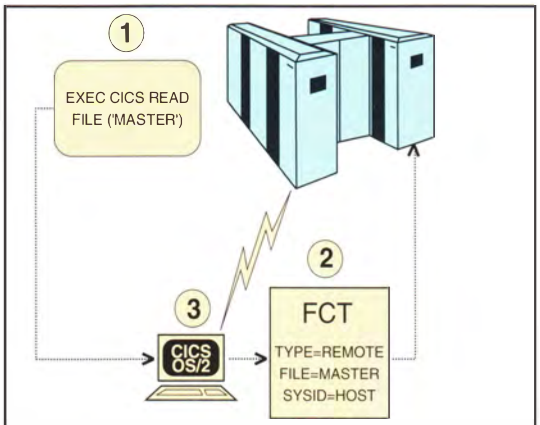
INTERCOMMUNICATIONS

Cooperative processing is fast becoming the data processing phrase of the 90's. In simple terms, cooperative processing is the ability to easily share data processing work across a single or multiple (alike or different) computer complex. Mainframe CICS users have been doing this for years through the use of CICS intercommunication facilities. (We'll spend more time with this topic then the others because it is here that the strength of CICS can be seen: the ability to harness the power of two very different computing platforms).

CICS intercommunication facilities allow two or more systems or regions to communicate and/or share terminals and other resources. The two modes of intercommunication are multiregion operation (MRO) (sometimes referred to as interregion communications, or IRC) and intersystem communications (ISC). For OS/2 programmers, a CICS region is defined as a single complete image of CICS with all its basic management facilities and functions.[4]

TRADITIONAL CICS REGIONS

As user applications are added the CICS control tables grow in size. Remember, it is through these tables that CICS knows about its resources. As the application grows, so do the control tables and so does the CICS region. Soon memory constraints (real and virtual) are reached. One way of relieving this memory constraint would be to continue to create independent CICS regions. This approach has two drawbacks. The first is that operating systems have limits, too. The second is that it can be difficult for end users to move between applications on the different systems. In most cases the user would have to logoff one system and logon to the other. This is because the same terminal can only be attached to one CICS region at a time. If an end user has applications on several different CICS regions the user would have to logoff of one region before they could logon to the other. This can be tedious.



1. CICS OS/2 Program issues file read command.
2. CICS OS/2 File Control Table has file defined as remote and has name of remote CICS system.
3. Read function is shipped for execution on file owning system.
NOTE: The CICS functions and table description for Figures 2, 3 and 5 have been simplified for this article. See the appropriate CICS documentation for detailed descriptions.

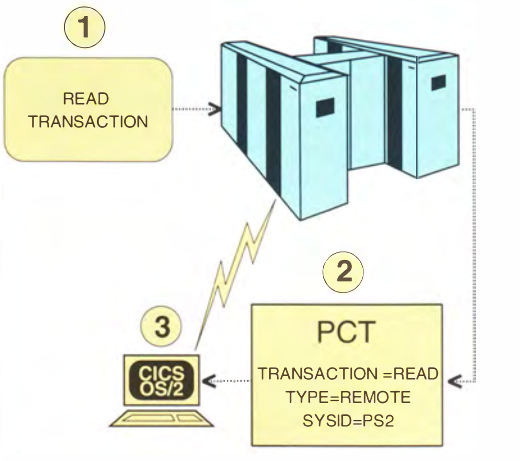
Figure 3. Function Shipping

SPLITTING REGIONS BY FUNCTION

With MRO and ISC a single large CICS region can be divided into two or more smaller regions along lines of function.

[4] A single mainframe operating system like *MVS* can manage large numbers of *CICS* regions on a single 3090 computer. In contrast you can only have one *CICS OS/2* executing on a single *PS/2*, but that's all that is needed!

CICS OS/2 makes it practical to develop programs that can run at either a mainframe or a PS/2.



1. Read Transaction is entered at a terminal attached to a host CICS.
2. CICS Program Control Table has transaction defined as remote and name of remote CICS system.
3. The transaction is routed to remote CICS for execution.

Figure 4. Transaction Routing

For example, one region could define all terminals, a second all applications, and a third all files. In CICS we like to use acronyms, so...the terminal region is called a TOR for Terminal Owning Region, the application region is called a AOR for Application Owning Region, and the data region is called a DOR (or FOR) for Data (File) Owning Region (see Figure 2).

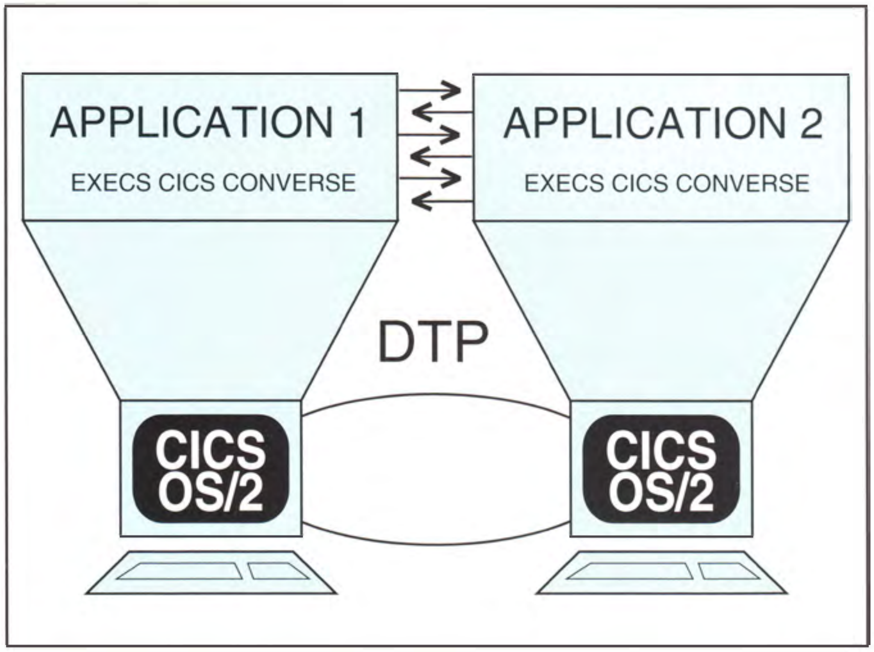


Figure 5. Distributed Transaction Processing

This means that each of these CICS regions now has room for application growth. However this does not mean that these regions are limited to a single function. An AOR can, and many times does, have terminals defined to it. An FOR can execute applications as well as have terminals defined.

How are these MRO regions created? It's all done with tables. The CICS system part is defined as normal. It is through the CICS control table that the resources used by applications are divided between regions. For example, in the table used to define transactions to CICS, the program control table (PTC), one parameter would tell CICS that the transaction executed on another system and a second parameter would tell CICS the name of the other system. In the terminal control table, TCT, we would define the other system and the communication links. The table definitions are made by the CICS system programmer and are transparent to the applications. An application program that is currently executing in a stand alone CICS region could be executing in an MRO system with no change to the program.

In simple terms, the multiregion option is for CICS-to-CICS communications within the same mainframe complex. Intersystem communications are for CICS-to-CICS links between processor complexes as well as interconnected CICSs on the same processor. Mainframe CICS-to-CICS OS/2, as well as CICS OS/2-to-CICS OS/2 communications use ISC. It is through these functions that CICS cooperative processing is as simple as coding a single command.

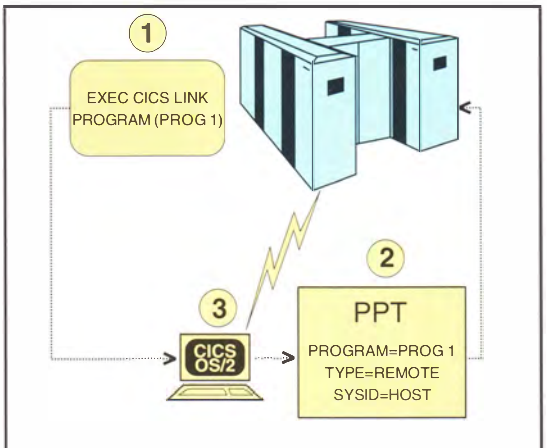
Now let's review the facilities that intercommunications offer.

Function Shipping (see Figure 3) allows command-level application programs in one CICS system to address resources in any connected CICS system. This means that when a program issues a file read command, the CICS file control program searches the file control table (FCT) for the file's entry. The entry indicates that the file is remote and provides the name of the remote CICS system. The file request is then passed to the intercommunications program and shipped to the file owning region. Once the data is retrieved or an error raised, the data or error code is returned to the originating CICS region and program. Data location is of no concern to the application program. For example, a mainframe CICS application could use function shipping to deposit sales quota and product inventory data at each PS/2 for the next business day.

Transaction Routing (see Figure 4) allows operators of terminals in one CICS region to run transactions in any connected CICS system. A terminal operator at a terminal connected to one CICS can enter a transaction code for a transaction without being aware of the location of the transaction. CICS will search the PCT for the transaction entry. The entry will indicate that the transaction is remote and the name of the remote system. CICS will route the request to the application owning region (AOR), and the transaction will run exactly as if the terminal were attached to the AOR. CICS handles all routing of requests and replies between the two systems. In general, transactions can be designed and coded without regard to the fact that the terminal is connected to another CICS system. For example, a mainframe CICS application could use transaction routing to start a transaction at each PS/2 to pick off end of business day volumes and send them up to the mainframe for central site processing.

Distributed Transaction Processing (see Figure 5) allows synchronous communications between transactions executing in different systems. This means that a session is acquired and held by two transactions for the period of a *conversation* between them. Because the transactions have exclusive use of the session, messages that pass between them as part of the conversation can be directly correlated, and each transaction can carry out processing that depends directly on the results of a previously performed stage of processing by the other. In other words DTP is a conversation controlled by the application developer. It is through the use of DTP that applications can be developed that will have one program executing on a CICS system and its partner on a non-CICS system. This could just as easily be programs on two CICS systems. DTP gives the application developer great control but it also requires a knowledge of APPC commands and the APIs of any non-CICS attached systems.

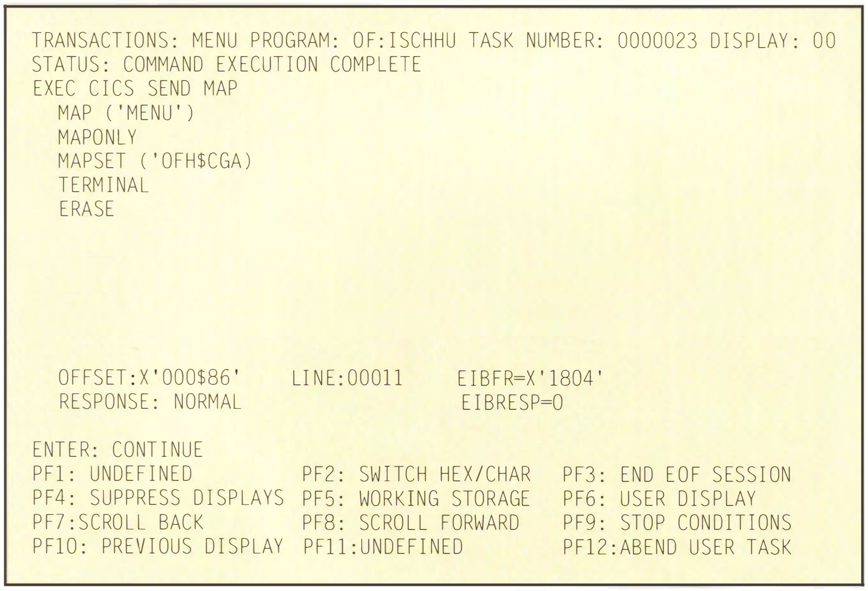
Distributed Program Link (see Figure 6) allows a command level application program in CICS OS/2 to link to a program in any connected CICS system. CICS program control management programs allow one application program to pass execution control to another. This is done as either a LINK function, where the second program returns control to the first; or as a transfer of control, where the second program gives control to another program (as a link or transfer) or to CICS. In mainframe CICS this passing of control could only be done between programs executing in the same CICS region. MRO or ISC systems could not be used. CICS OS/2 introduced the distributed program link (DPL) facility. When an application program executing on a CICS OS/2 system issues a LINK command, CICS searches the table that defines



1. CICS OS/2 Program issues LINK command.
2. CICS Program Properties Table has program defined as remote and name of remote CICS system.
3. Request is sent to the remote system and PROG 1 is executed. When PROG 1 ends, control is returned to the CICS OS/2 program.

Figure 6. Distributed Program Link

programs, the program properties table (PPT). The program entry indicates that the program is remote and returns the name of the remote system. DPL uses a COMMAREA to pass data to and from the linked system. A CICS OS/2 application could be designed to read and validate operator-entered data and use DPL to pass this data to another CICS (OS/2 or mainframe) system. By not having to send full screens of data, the load on the network is lessened. DPL does not support transfer control.



```
TRANSACTIONS: MENU PROGRAM: OF:ISCHHU TASK NUMBER: 0000023 DISPLAY: 00
STATUS: COMMAND EXECUTION COMPLETE
EXEC CICS SEND MAP
  MAP ('MENU')
  MAPONLY
  MAPSET ('OFH$CGA)
  TERMINAL
  ERASE






  OFFSET:X'000$86'    LINE:00011     EIBFR=X'1804'
  RESPONSE: NORMAL                    EIBRESP=0

ENTER: CONTINUE
PF1: UNDEFINED         PF2: SWITCH HEX/CHAR   PF3: END EOF SESSION
PF4: SUPPRESS DISPLAYS PF5: WORKING STORAGE   PF6: USER DISPLAY
PF7:SCROLL BACK        PF8: SCROLL FORWARD    PF9: STOP CONDITIONS
PF10: PREVIOUS DISPLAY PF11:UNDEFINED         PF12:ABEND USER TASK
```

Figure 7. Typical EDF Display

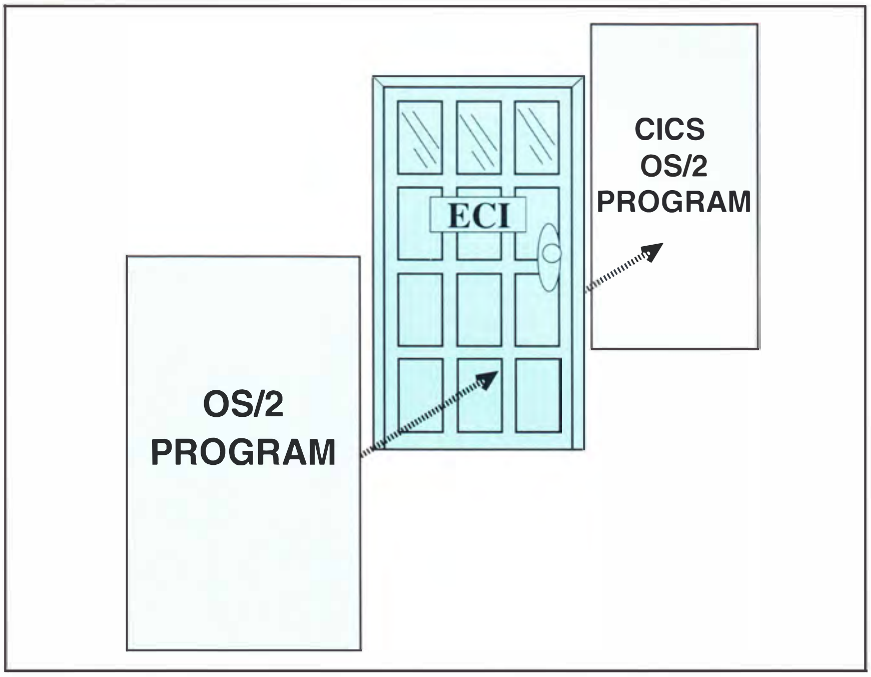


Figure 8. External Call Interface (ECI)

Function Shipping, Transaction Routing and Distributed Transaction Processing can be initiated from either mainframe or OS/2 systems. Distributed Program Links can be initiated from one CICS OS/2 application to another, or to a mainframe CICS, but not the other way around. This may not always be the case, since new functions introduced for one member of the CICS family are usually considered for other CICS products for the sake of compatibility and symmetry.

Other Facilities

We have only touched upon a few of the major facilities of CICS and CICS OS/2. CICS also provides data queueing facilities in transient data and temporary storage, a Security interface and user exit points. These and other facilities are documented in the CICS library. Languages supported on mainframe CICS are IBM/370 Assembler, COBOL, VS COBOL II, PL/I and C/370. CICS OS/2 language support is COBOL/2 and C/2. In general the source code of a program written in Command level CICS with either COBOL or C can be uploaded or downloaded between systems as source code. On either system CICS provides facilities that aid the programmer with program testing and debugging. One such tool is the Command Execute Diagnostic Facility (CEDF). CEDF enables you to test an application program online without modifying the program. CEDF intercepts execution of the application program before and after the execution of a CICS command (see Figure 7).

Another tool is the command level interpreter (CECI) which gives you the means of interacting with the CICS system. You can use CECI to enter CICS commands, check syntax, and execute these commands interactively at a CICS terminal. You can also use CECI in test situations; for example, to create test data.

CICS OS/2 ENHANCEMENTS

Most of the facilities reviewed so far are those that can be used by applications on either platform, but this doesn't mean that all CICS applications must be written for both. CICS can be used in many stand alone OS/2 applications that could exploit OS/2 facilities such as Presentation Manager or Dialog Manager. If mainframe data or processing is needed, the program could easily use CICS intercommunication facilities: function shipping, transaction routing, and program link.

Non-CICS OS/2 programs can also interface with CICS through External Call Interface (ECI) requests (see Figure 8). For example, you could develop an application with an EASEL front end and CICS for the data manager. The EASEL program would pass ECI data requests to the CICS program, where the data would be selected and returned to the EASEL program for display. The data could be local, on a LAN file server, or at a host. Wouldn't it be something to put together an application that could retrieve DB2 data and display it in spreadsheet format at the PS/2? With the CICS OS/2 ECI and a CICS cooperative processing facility like the distributed program link, this can be done.

APPLICATIONS ENABLING PLATFORM

In the title of this article we referred to CICS as an applications enabling platform. By definition, to *enable* is to provide the means or opportunity, to make possible, practical, or easy. All of this defines CICS OS/2 at the workstation. With CICS OS/2 we are provided with the means to develop applications that can use the power of both an IBM mainframe and an IBM PS/2. It makes it easy for an application at the PS/2 to

Current CICS Products:

CICS/MVS	Version 2.1.1 and 2.1.2
CICS/ESA	Version 3.1.1
CICS/DOS/VS	Version 1.7
CICS/VSE	Version 2.1
CICS/VM	Version 2.0
CICS OS/2	Version 1.20

General Availability Third Quarter 1991

CICS/ESA	Version 3.2

Figure 9. The Current CICS Family

CICS Publications - this is a partial list of publications:

CICS General Information	GC33-0155
Library Guide	GC33-0356
CICS/MVS Version 2.1.1	
Facilities and Planning	SC33-0504
Release Guide	SC33-0505
Application Programmer's Reference	SC33-0512
CICS/ESA Version 3.1.1	
Facilities and Planning	SC33-0654
Release Guide	SC33-0655
Application Programming Guide	SC33-0675
Application Programmer's Reference	SC33-0676
CICS/DOS/VS Version 1.7	
Facilities and Planning	SC33-0228
Release Guide	GC33-0130
Application Programmer's Reference	SC33-0077
CICS/VM Version 2.0	
General Information	GC33-0571
Systems Support and Administration	SC33-0573
CICS OS/2 Version 1.20	
System and Application Guide	SC33-0616

Figure 10. Partial List of IBM CICS Publications

access, with full integrity, mainframe data. It makes it possible to display this data using several different methods. CICS OS/2 makes it practical to develop programs that can run at either a mainframe or a PS/2. The programming skills to do this are the same. Now reread the opening development scenarios; how difficult would it be to develop these systems with CICS and CICS OS/2?

ACKNOWLEDGMENTS

My thanks to Bob Yelavich of the WSC in Dallas for keeping this article honest. ToJason McMahon of Hursley Park for his comments and help with the diagrams. ToTony Rummans, Louise Andres, and Cynthia Hobson of LA PS&U for giving me the time to write it.

Harry Starkman, *IBM United States M&S, Los Angeles Public Sector & Utilities Branch, 355 South Grand Avenue, Los Angeles, California, 90071. Mr. Starkman is an Advisory Systems Specialist in CICS. He has been with IBM for 2 years. Prior to IBM he spent 8 years as a CICS Systems Programmer. Mr. Starkman has a BA degree in Philosophy from The City College of New York.*

Application Enablers

Solving the Problem of Data Accessibility

by Sung Im

Software Publishing Corporation's InfoAlliance™, which was released in September of 1990, is the first product in the new PC software category known as "datasource integration." Datasource integration software integrates data from multiple sources and presents it through an intuitive graphical interface.

Within most organizations, essential information resides in many locations. It is generally kept on multiple platforms and in different databases associated with various software products.

In the past, there hasn't been a way to seamlessly integrate data from different sources and make it easily accessible to the user without MIS involvement.

InfoAlliance is a client-server product that enables the user to extract data from multiple sources, perform extensive analysis on the data, then present and communicate the results. The InfoAlliance user performs these three stages of information management in a smooth and intuitive manner and, therefore, can arrive at more timely and better-informed business decisions.

InfoAlliance allows users on a network to transparently access current data. This, and the common interface used by InfoAlliance, allows organizations to readily make the transition to new technologies, allowing them to adopt new software products without incurring the high cost of training and retraining users.

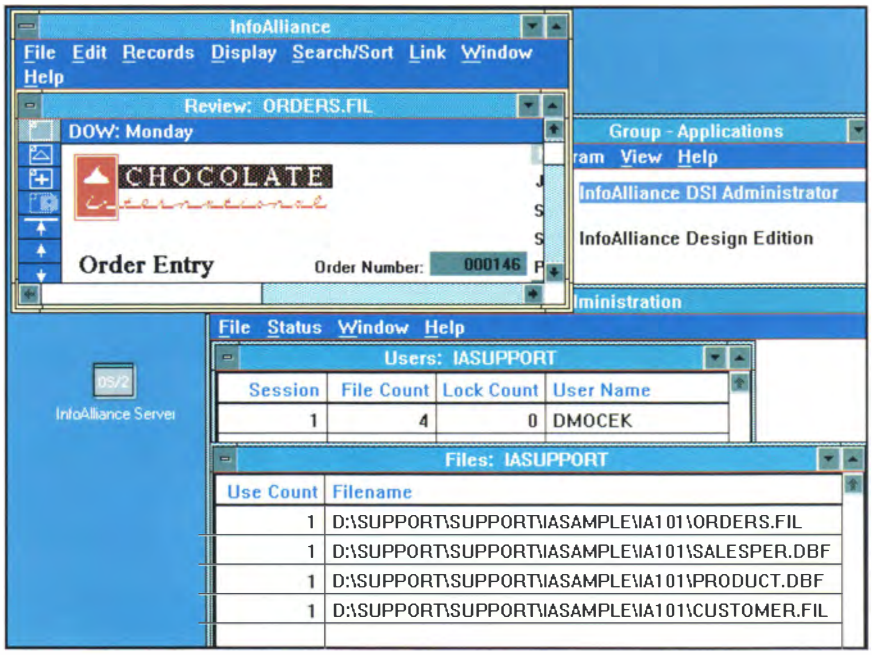
Product Summary

Name:	Software Publishing Corporation 1901 Landings Drive P.O. Box 7210 Mountain View, CA 94039-7210
Category:	Datasource Integrator
Description:	InfoAlliance integrates multiple data sources through a commom graphical information interface to allow viewing, generation of applications and reports.
Other Products:	Harvard Series, Professional Series, InfoAlliance products and other business productivity software.
Target Market:	MIS managers, application developers, business professionals, and PC/Micro/ Information center managers.
Platforms:	OS/2 PM, Client software available under Windows 3.0 in Calendar Q3, 1991
First Available:	September 1990
International:	Offices located in Canada, England, the Netherlands, Australia and Italy.
Suggested Memory:	8MB RAM
Price:	$995 – 1 client, 1 server $8,500 – 10 clients, 1 server $23,500 – 35 clients, 3 servers
Employees:	725
Sales Network:	InfoAlliance is sold through select Value Added Resellers (VARS) and Select Resellers.

Sung Im

For example, InfoAlliance currently provides a transparent way for corporations to migrate from Ashton-Tate dBASE to IBM OS/2 Extended Edition Database Manager (EE DBM).

Because InfoAlliance is based on advanced technologies, it contains the data security, integrity and administrative features necessary to protect and manage critical data across the network. Since users' view of data is unaffected by data source location, servers and data sources can easily be added and repositioned for optimum network performance without disrupting work.



Order Entry

Order Number: 000146

Session	File Count	Lock Count	User Name
1	4	0	DMOCEK

Use Count	Filename
1	D:\SUPPORT\SUPPORT\IASAMPLE\IA101\ORDERS.FIL
1	D:\SUPPORT\SUPPORT\IASAMPLE\IA101\SALESPER.DBF
1	D:\SUPPORT\SUPPORT\IASAMPLE\IA101\PRODUCT.DBF
1	D:\SUPPORT\SUPPORT\IASAMPLE\IA101\CUSTOMER.FIL

Figure 1. Example of Server Status Utility

COLLECTION OF DATA

Until now, collection of data from various sources has been a technically difficult and cumbersome process. Often, business professionals exchange data manually. That is, one department produces a hardcopy report, which another department re-enters into its system. Magnetic media, such as tapes and diskettes, are used to eliminate the need for reentry, but the process of downloading and uploading is still very cumbersome.

With the emergence of advanced operating systems like OS/2 and enabling technologies like Communications Manager and LAN Server, the business enterprise can now be physically connected as a whole; PCs on desktops can exchange E-mail messages with host mainframes and can even download mainframe data.

InfoAlliance takes full advantage of these enabling technologies to provide easy, yet controlled, access among computers throughout the business organization. Specifically, InfoAlliance provides easy access to data by implementing the following features:

- **Platform transparency :** InfoAlliance 1.0 client and server modules run on OS/2. InfoAlliance 1.1 client runs on Windows 3.0 and OS/2 Presentation Manager (PM). Because InfoAlliance is based on client-server architecture, users can better fit InfoAlliance into their hardware and operating system configuration. For example, the administrator may choose to purchase InfoAlliance OS/2 Clients for *power users*, while providing InfoAlliance Windows for *access users*. The administrator may also install the InfoAlliance 1.0 server on a fast machine with greater disk space.

- **Location transparency:** Location transparency enables the user to access data without knowing which servers the data sources are on or where the servers are located. Of course, on rare occasions when the data location needs to be identified, the user can easily specify site names. Location transparency removes the user's need to know how the network is configured. Further, it reduces network administration effort dramatically. In fact, if the administrator decides to add another client-server machine, the administrator can connect the new machine to the network, install InfoAlliance, and start InfoAlliance with a minimum of configuration work.

- **Multi-server architecture:** The InfoAlliance client process can communicate with multiple servers at the same time. The user is not required to know to which servers the user is connected. (See Figure 1.)

Because InfoAlliance performs the access function transparently, the user does not need to be aware of the format in which the data is stored. Users operate EE Database

Manager data using the same search and sort methods are used with dBASE files and other types of data. (See Figure 2.)

Transparent access allows users to protect their investment in popular database management systems (DBMS's). Existing EE and dBASE applications, for example, can run along with InfoAlliance. Different levels of transparency enable the InfoAlliance user to design a form in which a portion of data is accessed from IBM EE Database Manager on machine A while another portion is accessed from dBASE files on machine B.

ANALYSIS

InfoAlliance enables the user to perform extensive analysis on the accessed data. InfoAlliance supports non-stored fields that are calculated results of other fields on the form. These results can then be used to perform additional calculations.

If data needs to be extracted from multiple files, one can use the link facility in InfoAlliance. If an invoice form requires customer information, the designer of the invoice form can define a link in the invoice form to the customer file, using, for example, the customer identification number. Then, the customer name, address, and telephone number can be retrieved from the customer file through the use of the calculation facility.

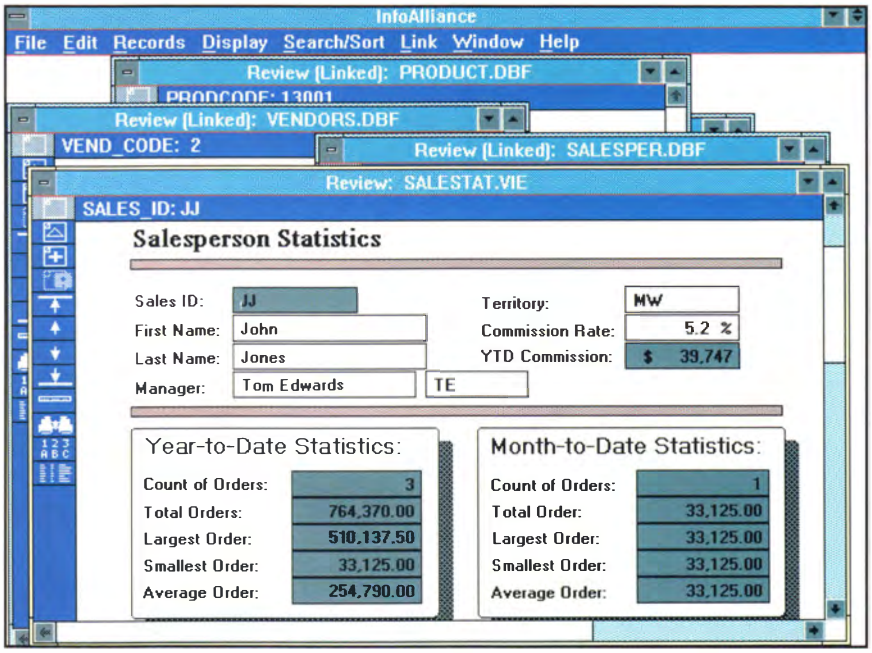
InfoAlliance performs sophisticated dependency analysis to perform calculations and look-ups on an as-needed basis. Because unnecessary calculations are not carried out, the user can perform *what-if* analysis interactively with optimized performance. In the previous invoice form example, the customer name, address, and telephone number are not refreshed as long as the customer identification number is left intact.

Furthermore, a rich set of file-wide and form-wide aggregates are available through which the user performs summarization analysis. Aggregates can be used to sum, count, average and find maximums and minimums for the fields defined by the user. A comprehensive set of built-in functions is supported as well.

PRESENTATION

InfoAlliance 1.0 takes full advantage of OS/2 PM. The graphical user interface and careful selection of icons and menus enable the user to manage information in an intuitive manner. InfoAlliance complies with Common User Access (CUA); the user who learns a CUA application can operate InfoAlliance with little or no training.

InfoAlliance's Form Design Editor supports a comprehensive set of drawing tools. Using these tools, the designer can, in an object-oriented manner, design a presentable business form. Color and graphical attribute support can be used to accent specific areas of the form. Each field in the form can be assigned text attributes, including typeface and font. (See Figure 3.)


InfoAlliance
File Edit Records Display Search/Sort Link Window Help
Review (Linked): PRODUCT.DBF
PRODCODE: 13001
Review (Linked): VENDORS.DBF
VEND_CODE: 2
Review (Linked): SALESPER.DBF
Review: SALESTAT.VIE
SALES_ID: JJ
123 ABC

Salesperson Statistics

Sales ID:	JJ	Territory:	MW
First Name:	John	Commission Rate:	5.2 %
Last Name:	Jones	YTD Commission:	$ 39,747
Manager:	Tom Edwards TE		

Year-to-Date Statistics:		Month-to-Date Statistics:	
Count of Orders:	3	Count of Orders:	1
Total Orders:	764,370.00	Total Order:	33,125.00
Largest Order:	510,137.50	Largest Order:	33,125.00
Smallest Order:	33,125.00	Smallest Order:	33,125.00
Average Order:	254,790.00	Average Order:	33,125.00

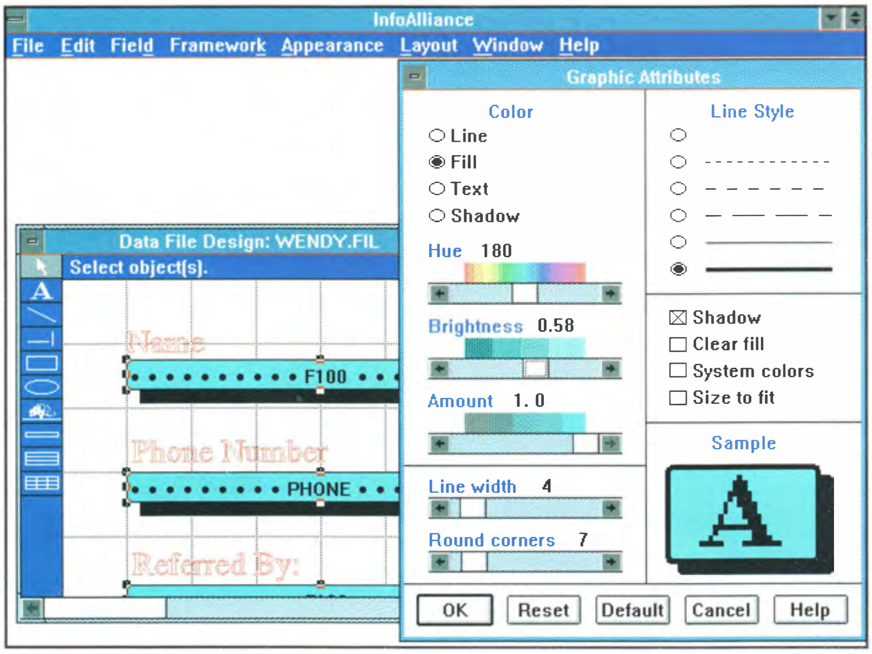
Figure 2. Example of Datasource Integrator

FRONT END TOOL — FORMS SUBSYSTEM

InfoAlliance uses business forms as the primary interface metaphor through which information is managed. The forms subsystem enables the user to design business forms and use them for analysis and presentation. InfoAlliance's Form Design Editor looks similar to a drawing program. A major difference, however, is that InfoAlliance allows the user to *draw* (define) data objects as well as graphical objects.

The user is given a set of easy-to-manage dialog boxes through which the data definitions of data objects can be modified.

InfoAlliance also supports scanned images. Images brought in through a scanner can be *back-dropped* into an electronic form. That is, these images of existing paper forms can be used as the canvas onto which data objects are drawn. (See Figure 4.)



Color
○ Line
● Fill
○ Text
○ Shadow

Hue 180

Brightness 0.58

Amount 1. 0

Line width 4

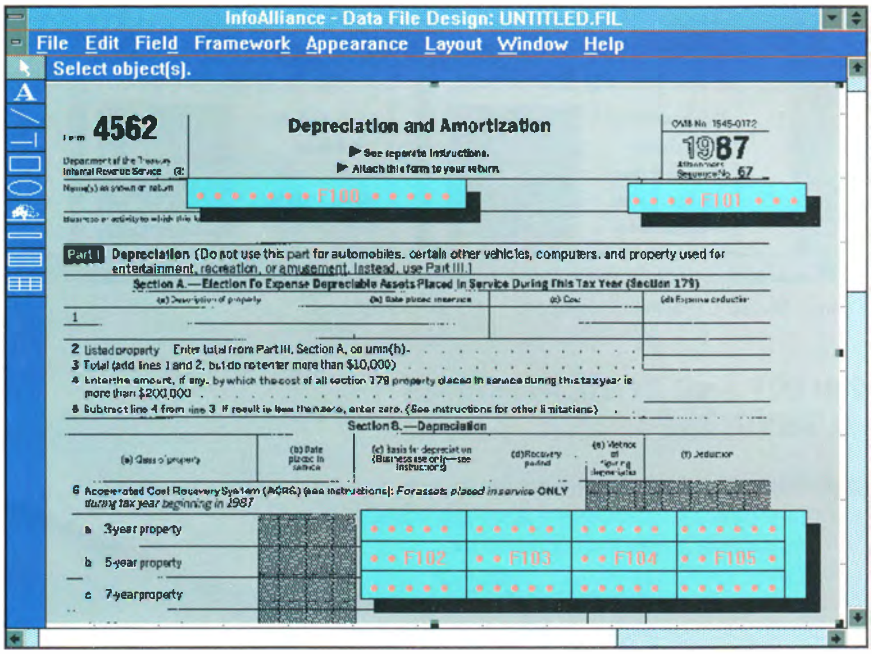
Round corners 7

Line Style

☒ Shadow
□ Clear fill
□ System colors
□ Size to fit

Sample

Figure 3. Example of Graphic Application Generator


InfoAlliance - Data File Design: UNTITLED.FIL
File Edit Field Framework Appearance Layout Window Help
Select object(s).
F100
F101
F102
F103
F104
F105

Form 4562

Department of the Treasury
Internal Revenue Service

Depreciation and Amortization

▶ See separate instructions.
▶ Attach this form to your return.

OMB No. 1545-0172

1987

Attachment
Sequence No. 67

Name(s) as shown on return

Business or activity to which this

Part I Depreciation (Do not use this part for automobiles, certain other vehicles, computers, and property used for entertainment, recreation, or amusement. Instead, use Part III.)

Section A.—Election To Expense Depreciable Assets Placed in Service During This Tax Year (Section 179)

(a) Description of property	(b) Date placed in service	(c) Cost	(d) Expense deduction
1			

2 Listed property. Enter total from Part III, Section A, column (h)

3 Total (add lines 1 and 2, but do not enter more than $10,000)

4 Enter the amount, if any, by which the cost of all section 179 property placed in service during this tax year is more than $200,000

5 Subtract line 4 from line 3. If result is less than zero, enter zero. (See instructions for other limitations)

Section B.—Depreciation

(a) Class of property	(b) Date placed in service	(c) Basis for depreciation (Business use only—see instructions)	(d) Recovery period	(e) Method of figuring depreciation	(f) Deduction
6 Accelerated Cost Recovery System (ACRS) (see instructions): *For assets placed in service ONLY during tax year beginning in 1987*					
a 3-year property					
b 5-year property					
c 7-year property					

Figure 4. Example of Background Form

The Form Design Editor manages other data definition objects, such as indices for optimizing search and sort performance, links for defining relationships among files, and privileges for security management.

InfoAlliance offers a wide range of security and access modes to provide a high level of security to both PC users and information managers. Data security is ensured with seven levels of user access, which can be set at the file, directory and server levels, depending upon the needs of the organization and the user.

InfoAlliance provides standard methods of verifying data and protecting its integrity including field validation formulas, full transaction commit control and rollback, with a two-phase commit protocol for distributed updates.

Once the form is designed, the user manages information easily and quickly through the form; data manipulation, entry and retrieval are done through the form: searching is done by giving examples of the search result in a blank form, and sorting is done by clicking on the fields that are to be sorted.

FRONT END TOOL — REPORT SUBSYSTEM

InfoAlliance is equipped with two report writing systems: Quick Reports and Custom Reports. Quick Reports is ideal for columnar reports. The Quick Reports design editor uses a *point-and-click* method. This feature allows users to set up criteria to quickly select and sort data. A blank form is presented on the design editor; the user double clicks on field (data) objects to include them in the report. Quick Reports also enables users to preview reports prior to printing.

Custom Reports, on the other hand, is used for more complex documents. Graphical objects can be included in the report, data objects can be placed anywhere, and complex grouping hierarchy can be specified. Users can select from an array of graphic capabilities including boxes and lines, scanned images, colors, and fonts to customize their reports. (See Figure 5.)

FRONT END TOOL— APPLICATION GENERATOR

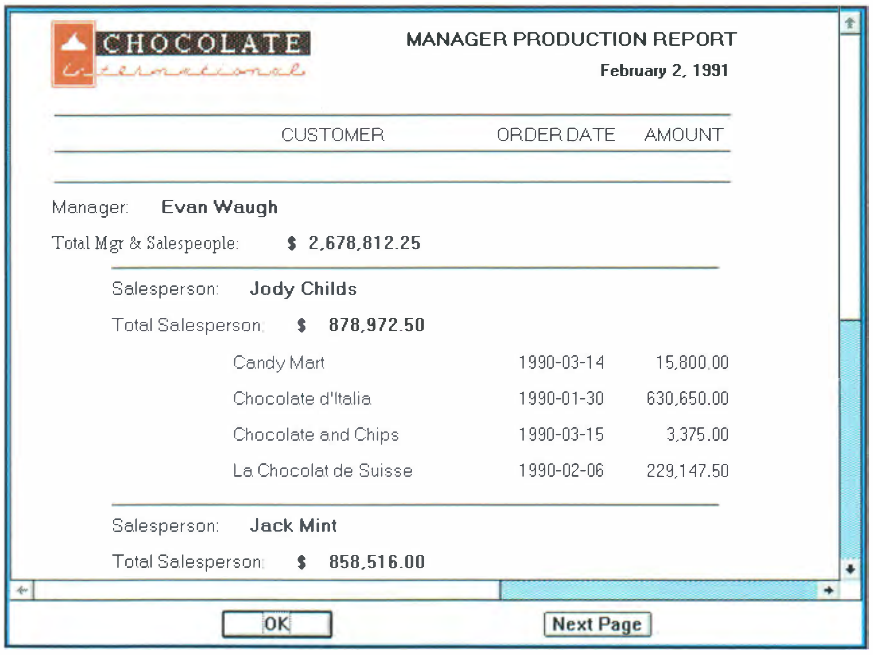
InfoAlliance's Application Generator allows the user to tie forms and reports into a customized application. The user defines a menu. To each menu, one or more menu items are assigned. Each menu item can then be assigned a number of *actions.*

This feature allows users to use their own terminology to create menu names and menu items. It also helps them produce consistent results by creating a set of pre-defined procedures for each menu item.

A simple menu item may open a specific file, perform a search, and review records that meet the search criteria. In this case, the user simply chooses these actions from the pre-defined list of actions.

When the search action is added to the menu item, the search form for the file is brought up. The user (application designer) can specify the default search.

The Application Generator also enables the user to test the application interactively while designing the application. The user has an option to specify that changes made during the test are temporary; in this case, all data manipulation operations performed during the application test are backed out when the user finishes the test. This allows the designer to develop an application interactively and incrementally.



CHOCOLATE International

MANAGER PRODUCTION REPORT

February 2, 1991

	CUSTOMER	ORDER DATE	AMOUNT
Manager: **Evan Waugh**			
Total Mgr & Salespeople: **$ 2,678,812.25**			
Salesperson: **Jody Childs**			
Total Salesperson: **$ 878,972.50**			
	Candy Mart	1990-03-14	15,800.00
	Chocolate d'Italia	1990-01-30	630,650.00
	Chocolate and Chips	1990-03-15	3,375.00
	La Chocolat de Suisse	1990-02-06	229,147.50
Salesperson: **Jack Mint**			
Total Salesperson: **$ 858,516.00**			

Figure 5. A Custom Report

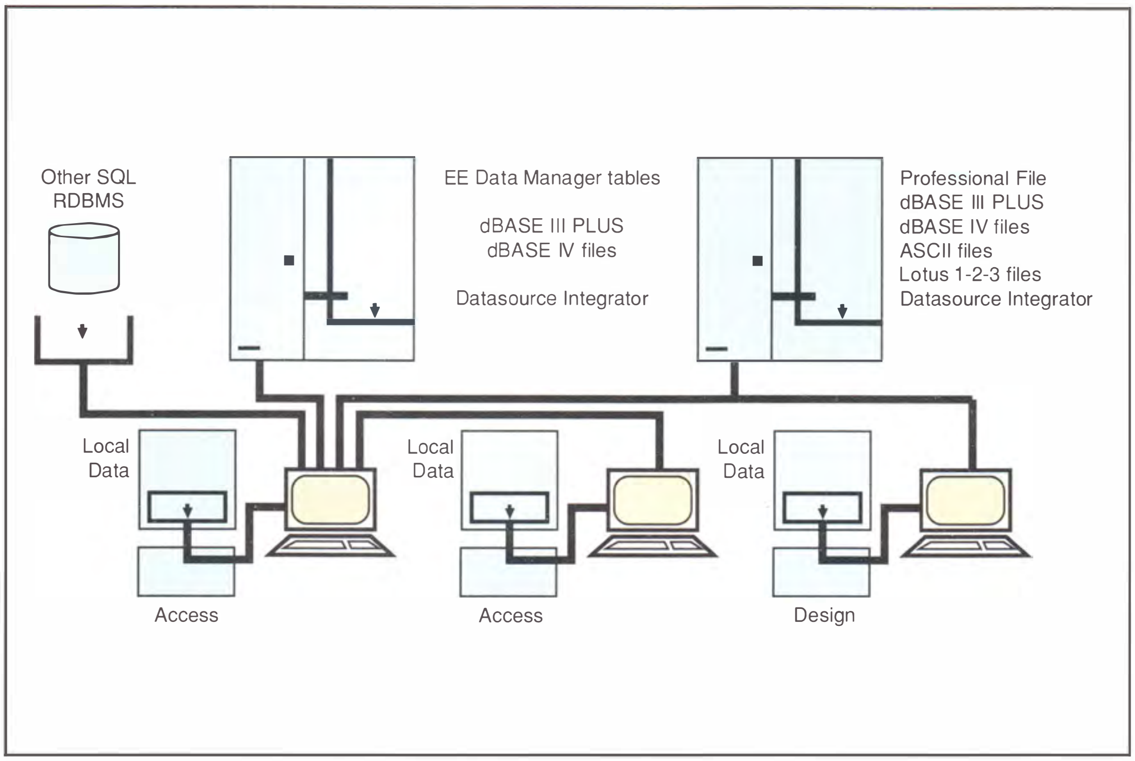


Figure 6. The InfoAlliance Client Server Allows the Integration of Data from Multiple Sources

HIGHLIGHTS OF INTERNAL WORKINGS

As mentioned before, InfoAlliance is based on the client-server architecture. The server may also be configured to run on the same machine as the client process. If the server runs on the same machine as the client, the two processors communicate with each other through OS/2's shared memory facility. (See Figure 6).

If the server and client are located on different machines, they communicate through the NetBIOS facility of LAN/Manager or IBM EE Communications Manager. Code specific to data communication is isolated so that support for other protocols can be added easily.

InfoAlliance complies with Common User Access (CUA); the user who learns a CUA application can operate InfoAlliance with little or no training.

The front-end tools are all based on a logical layer called the Presentation Layer, which hides the details of the underlying graphical user interface (GUI) system. In the case of OS/2, the GUI is Presentation Manager (PM). Furthermore, the presentation layer API calls are at a much higher level than PM. This enables the porting of InfoAlliance front-end tools to other GUI environments. For example, the major effort to port the InfoAlliance client tools to Windows 3.0 is in porting the presentation layer to Windows 3.0.

InfoAlliance's connectivity to various data formats can be categorized into two strategies; one for the DBMSs that have open application program interfaces APIs and the other for those that do not have open APIs.

IBM EE Database Manager belongs to the former, while dBASE belongs to the latter group. In either case, the connectivity code resides in the InfoAlliance server. The InfoAlliance client makes a request to the server to perform a Data Definition Language (DDL) or Data Manipulation Language (DML) operation. The InfoAlliance server determines the data format destination for the request.

If the request is for a DBMS with open APIs, the request is sent to a dedicated process that services the user. In the case of IBM EE DM, this service process is written in IBM SQL/C.

If two InfoAlliance users access IBM EE DM concurrently, two transparency service processes will be initialized. The concurrency control between these two processes and other DM applications is governed by EE DM.

For DBMSs without open APIs (as is the case with dBASE), the InfoAlliance server manages the files itself. The concurrency control on these files is governed by the InfoAlliance server.

Because the connectivity code resides within the server, users can perform queries that span multiple machines on EE DM and dBASE at the same time. For example, one can create and define an invoice form in which the invoice master data comes from EE DM on one server, customer data from a dBASE file on another server, and the part information comes from EE on yet another server.

CONCLUSION

As enabling technologies such as OS/2, EE Database Manager, and EE Communications Manager emerge, the user expects a higher degree of usability, connectivity, integration, and power in application software.

Software Publishing Corporation's InfoAlliance is one of the first products to translate these technology advancements into customer benefits. InfoAlliance's solid and extensible architecture will enable Software Publishing Corporation to offer wider connectivity and a variety of front ends through which users can more readily access, analyze, and communicate information. This will, in turn, enable users to make better business decisions.

Sung Im, *Software Publishing Corporation, 1901 Landings Drive, P.O. Box 7210, Mountain View, CA, 94043-7210. Mr. Im is Section Manager of System Services, Information Access Division of Software Publishing Corporation. He has managed a leading development team effort for the new InfoAlliance product. Mr. Im has a strong database, data communication and OS technology background, and holds a BA from Harvard University.*

PIPES Platform™– A Distributed Computing System

by Peter Tait

THE PROBLEM

Businesses need applications. Some of these applications are *mission-critical*—activities that will have a direct impact on the success or failure of the enterprise, and corporations will only deploy these applications on platforms that are robust enough to provide reliable service.

Traditionally, these platforms have been mainframe and minicomputer systems, for both performance and reliability reasons. The rapid evolution of personal computing systems, which may now boast better than 10 MIPS on the desktop, has dramatically altered the balance of raw CPU power away from host systems.

This has presented corporations with a dilemma: on the one hand, the possibility of distributing application processing across the vast CPU resource of the installed PCs, and on the other, the problem of satisfying the business requirement that application platforms be robust enough for mission-critical functions.

ONE SOLUTION

Ideally, the level of service required to support robust distributed applications would be provided by the operating system, allowing the programmer to rely on persistent operating system services as the providers of communications functions. Since it is extremely unlikely that a single operating system providing these functions will replace the installed base of non-distributed operating systems within the next five to ten years, this functionality must be provided by enhancing these existing systems.

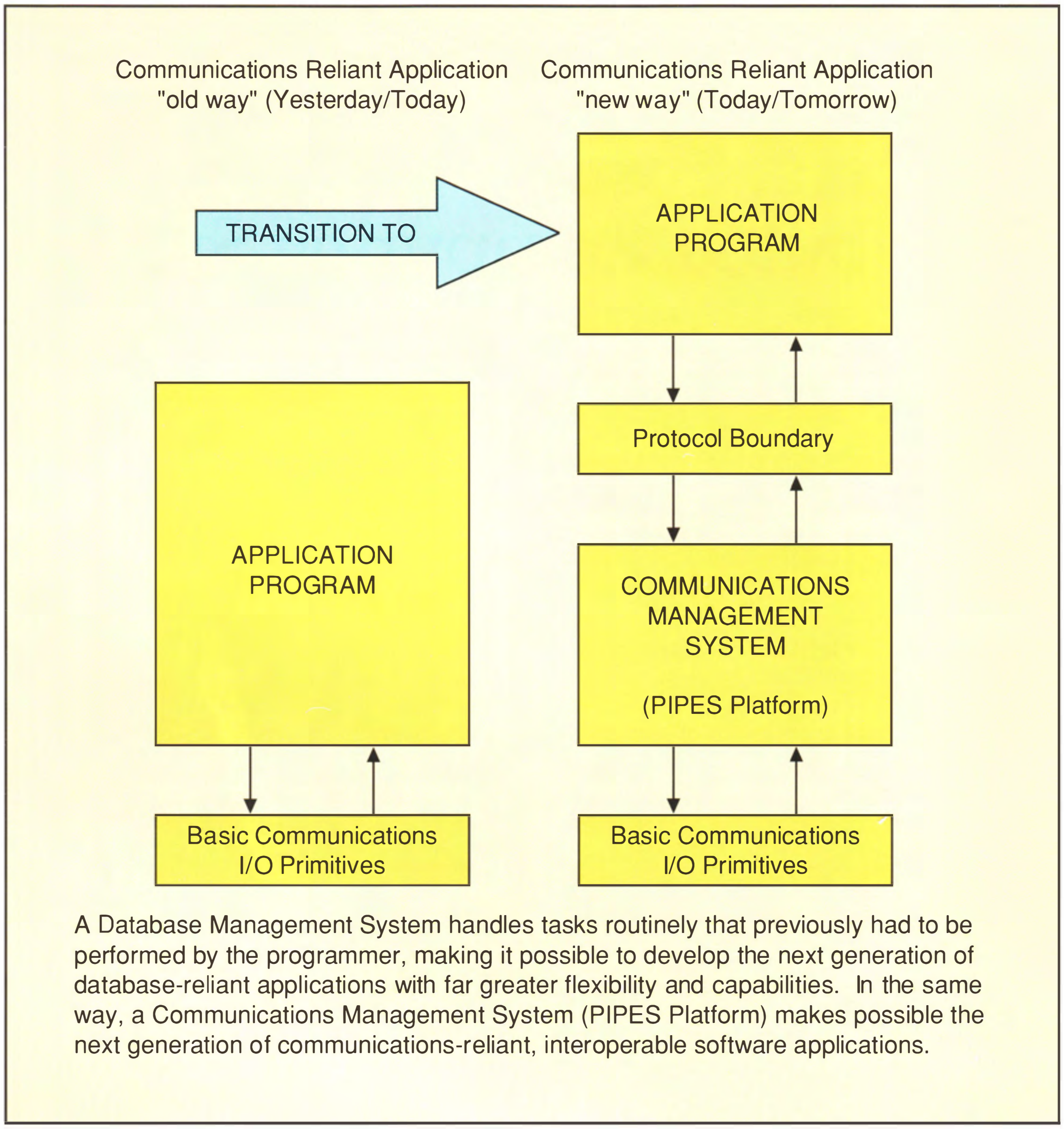
PIPES Platform™ Base System (v 3.0) from PeerLogic, Inc. of San Francisco, is a communications management system that is designed to solve this problem (see Figure 1). PIPES Platform is an implementation of a message-passing architecture that is notably easy to use. It provides application programmers with a high-performance process-to-process connection, using function calls that are very similar to those used for file I/O. All of the technology issues associated with the building of that connection, such as locating the processes, routing through the network and recovering from errors, are handled by PIPES. For example, PIPES deals with any recoverable communications error without even notifying the application.

Since the library of PIPES function calls is the only element of the system bound to the application, PIPES is able to locate and use transport protocols at runtime. Applications written for the NetBIOS or IPX environments today, for example, will run unchanged over OSI stacks as these become available and are supported by PeerLogic.

This makes applications developed using PIPES Platform inherently portable. As the product is ported to different environments, it will become a consistent communications service, adding an element of distributed operating system functionality to existing systems.

Peter Tait

PIPES deals with any recoverable communications error without even notifying the application.


Communications Reliant Application
"old way" (Yesterday/Today)
Communications Reliant Application
"new way" (Today/Tomorrow)
TRANSITION TO
APPLICATION PROGRAM
Protocol Boundary
APPLICATION PROGRAM
COMMUNICATIONS MANAGEMENT SYSTEM
(PIPES Platform)
Basic Communications I/O Primitives
Basic Communications I/O Primitives

A Database Management System handles tasks routinely that previously had to be performed by the programmer, making it possible to develop the next generation of database-reliant applications with far greater flexibility and capabilities. In the same way, a Communications Management System (PIPES Platform) makes possible the next generation of communications-reliant, interoperable software applications.

Figure 1. PIPES Platform – A Communications and Resource Management System

Since there is no single repository for this resource information, there is also no single point of failure for the PIPES Logical Network.

THE PIPES LOGICAL NETWORK

The PIPES kernel, through the PIPES Application Programmer's Interface, presents the programmer with a purely logical view of an arbitrarily complex physical network. This logical view is completely independent of the physical network; details such as topologies, routers and transport protocols are invisible to the programmer.

Definitions:

- All PIPES-based communication involves *messages* sent on *sessions* to and from *resources.*
- A *resource* is an object added to the PIPES Logical Network with which other processes can converse.
- A *message* is simply a buffer in memory which is pointed to in the send and receive message calls.

- A *session* is a single logical connection between processes, one of which has added a resource to the network.

- *Messages* sent by either end of the session will be received in sequence at the other end.

- *Processes* that want to be contacted by other processes must add one or more resources to the network, using the AddPiResource function.

- *Processes* wanting to communicate with a resource use the AllocPiSession function to establish the *pipe* between them, and the SendPiMessage and RecvPiMessage calls to communicate.

The PIPES kernels in the network, acting together, form the PIPES nested distributed operating system, so-called because PIPES adds distributed operating system functionality to operating systems that do not provide these services.

The Hierarchical Name Space

In order to handle the problem of resource location in extremely large networks, PIPES uses a hierarchical naming scheme to provide the logical view of the network. The fully qualified path to a resource consists of *class @ resource @ application @ machine@ area @ group @ domain @ network.*

The application has control only over the application, resource and class fields of resources it adds, with the kernel supplying the remainder of the path. When adding a resource to the network, an application is able to specify the scope of its availability; the resource can be limited to use by applications in the same area, group, etc.

When searching for resources, an application is also able to use wildcard characters to specify the scope of the search. For example, the application may specify a search for resources of a particular class, published by a particular application, located in any area within its group, by passing *class_name @ * @ application_name @ * @ * @ group_name @ domain_name @ network_name* to the FindPiResource function. Names must be unique for each level of the hierarchy: machine names within an area or area names within a group, for example.
At each level of the hierarchy (i.e., area, group, domain and network), one PIPES kernel will take on a privileged status and act as a *manager* for that portion of the PIPES Logical Network. This manager kernel is identical to all other kernels; the only difference is the function it is performing at a particular time.

In the event of a failure of a kernel acting as a manager, or its normal removal from the network, another kernel will automatically take over the responsibilities of the failed kernel, without user intervention or knowledge of the fault. For more information, see **Privileged Nodes**.

Following are descriptions of the hierarchical organization of the name space:

Network

A network consists of a number of domains. The kernels making up the network are the managers of the domains within that network. At any time, one of the Domain Managers will have taken on a privileged status and also be functioning as the Network Manager.

Since there is no single repository for resource information, there is also no single point of failure for the PIPES Logical Network.

The Network Manager is responsible for managing the interaction between its network and other networks in the hierarchical name space. This kernel will know the location of other Network Managers. In the event that the Network Manager fails, or is removed from the network, another member of the network (another Domain Manager) will automatically assume the responsibilities of the Network Manager.

Domain

A domain consists of a number of groups. The kernels making up the domain are the Managers of the groups within that domain. At any time, one of the Group Managers is also functioning as a Domain Manager.

The Domain Manager is responsible for managing the interaction between the group and the rest of the hierarchical name space. This kernel will know the location of the

The key to the PIPES Logical Network is the concept of the Resource, a named piece of executable code with which other programs can communicate.

Network Manager, since it is also a member of the network.

In the event that the Domain Manager fails, or is removed from the network, another member of the domain (another Group Manager) will automatically assume the responsibilities of the Domain Manager.

Group

A group consists of a number of areas. The kernels making up the group are the Managers of the areas within that group. At any time, one of the Area Managers is also functioning as a Group Manager.

The Group Manager is responsible for managing the interaction between the group and the rest of the hierarchical name space. This kernel will know the location of the Domain Manager, since it is a member of the domain. In the event that the Group Manager fails, or is removed from the network, another member of the group (another Area Manager) will automatically assume the responsibilities of the Group Manager.

Area

An area consists of a number of machines. Within an area, all resource information is fully replicated. Whenever an application adds a resource to the network, knowledge of that resource is replicated around the area. At any time, one of the kernels in the area will have taken on a privileged status and be functioning as the Area Manager.

The Area Manager is responsible for managing the interaction between the area and the rest of the hierarchical name space. This kernel will know the location of the Group Manager, since it is also a member of the group. In the event that the Area Manager fails, or is removed from the network, another kernel will automatically assume the responsibilities of the Area Manager.

Machine

Each node in the PIPES network is given a name, and that name must be unique within the area. This name, and information about the area, group, domain and network to which the machine belongs, is supplied by the user of the machine via a configuration file.

Thus, each PIPES kernel is aware of the manager station above it in the hierarchy, and if it is itself a manager, it also knows of the other managers at the same level, the manager above it, and the managers of the level below it.

A Group Manager, for example, will know of the Domain Manager, the managers of the other groups in the domain, and the managers of the areas within its own group. In order to be a manager at a particular level, a kernel must be a manager at all lower levels. Thus, the Network Manager is also a Domain Manager, a Group Manager and an Area Manager.

RESOURCES

The key to the PIPES Logical Network is the concept of the Resource, a named piece of executable code with which other programs can communicate. By definition, a resource is an abstract representation of a user-written program. A user can define the following characteristics of a resource:

Name

A resource name is represented by six sets: Resource name, Application name, Machine name (in which the application is executing), Area name (in which the machine is located), Group Name, Domain name and Network name. Please note that:

- An Application is a set of Resources.
- A Machine is a set of Applications.
- An Area is a set of Machines.
- A Group is a set of Areas.
- A Domain is a set of Groups.
- A Network is a set of Domains.

A set in this context might be a Null Set. A resource will have the following attributes:

- resource type
- resource class
- resource sub-class
- resource scope
- resource life

Class

The resource class and resource type indicate the class and the type of resource, respectively. Resource classes can be considered as overlapping sets of the subclasses. Types are unique sets of resources. Subclasses are overlapping sets of resources.

The resource scope defines the visibility of the resource. We will discuss the resource scope in the context of the PIPES Logical Network. The resource life indicates how long this resource is going to last with its given attributes at the current physical address. These parameters will be used to keep track of *transient resources.*

Information regarding resources with infinite life can be stored in a non-volatile storage of users.

PRIVILEGED NODES

A user may specify or designate a machine as a *manager machine.* A manager machine is responsible for the following:

- Routing Service
- Login Service
- Naming Service

A new Manager Machine will always be selected by PIPES in the event that the current manager goes down, even if a manager has been designated by the user. Users can designate multiple managers so that, in the case of failure of a manager machine, the newly selected manager will be a user-designated Manager Machine.

At any given instant there will only be one Manager Machine in an Area. A Manager Machine will be responsible for the construction of an Area as described below:

Construction of an Area

Every machine that is brought up over a physical network broadcasts a login message over the network. This message contains the name of the Machine as well as the name of the Area in which this Machine intends to login. If that Area already exists, the Machine is integrated into that area. Otherwise, the Machine logs in as a new Area. A user may specify a Machine as the Manager Machine. If a Manager Machine is not specified, PIPES will select a Machine as a Manager Machine.

The Manager will be responsible for logging in other Machines and making sure that the name of every Machine is unique within the Area.

Route Caching

Once a route to a particular resource within the PIPES Logical Network has been discovered—as a result of a FindPiResource call issued by an application— that information is cached within the local kernel. This caching speeds the locating of often-used resources.

The interface consists of 13 functions, callable from C language programs.

Driving Multiple Transports

The PIPES kernel, via the Context Bridge Layer, is able to drive multiple network protocol stacks simultaneously, and route PIPES-based conversation messages from one protocol to another. This means that applications using PIPES Platform do not require a homogeneous communications network in order to function, nor does the programmer need to specify the supported protocols during application development.

For example, an application running on a PC using the Novell NETWARE DOS client software (i.e., providing an IPX interface to PIPES) could be in peer session with its opposite number, running in another workstation using the DECnet protocols, even though their communications are routed via LU6.2 over a corporate SNA network.

THE PROGRAMMER'S PERSPECTIVE

The Distributed Application Architecture

Before even beginning to write applications that will be distributed across a network, the developer will need an architectural model, or a road map, to design these applications. Whether the application is to be client-server or peer-to-peer, it will be necessary for the processes making up the application to communicate with each other.

The communications solution adopted should provide the greatest possible flexibility for the programmer, allowing for fast, reliable movement of information from one process to another.

In other words, the architecture chosen by the developer should suit the needs of the application, rather than be dictated by the underlying communications technology.

PeerLogic believes that the only way to implement such a system is to use message-passing as the means of communication between processes. Although other systems, such as Remote Procedure Call technology, have been popular, we feel that they are not flexible or robust enough to satisfy developer requirements for the next generation of distributed applications.

In our opinion, PIPES Platform provides a high-performance message-passing scheme, allowing the programmer to rapidly build and deploy communications-reliant applications. The calls to the PIPES system are both asynchronous and non-blocking, executing *notification routines* supplied by the programmer when asynchronous events are completed.

Using PIPES

The PIPES Application Program Interface (PAPI) is the access method used to take advantage of the services offered by the PIPES kernel. The interface consists of 13 functions, callable from C language programs.

The list of functions can be subdivided into five categories: API Control, Resource Management, Session Management, Message Passing and Process Control.

API CONTROL

InitPiAPI()

InitPiAPI() initializes PAPI for the calling application. InitPiAPI must be called by a process before any other PAPI routine and it initializes the run-time environment, acquires system resources needed to communicate with the PIPES kernel, and identifies the process to PIPES as a new PAPI agent.

A process calling InitPiAPI identifies itself with a (host) unique application name. Several instances of a single program may become PAPI agents, provided that they use unique application names. Upon successful completion, InitPiAPI provides the caller with a fully qualified path name, which is a globally unique identification of the PAPI agent. InitPiAPI is always synchronous.

TermPiAPI()

TermPiAPI() terminates PAPI for the calling application. TermPiAPI should be called after all PAPI operations are concluded. It will release the system resources that are no longer needed, and will inform the PIPES kernel that the PAPI agent is terminating. After calling TermPiAPI, a process may not make any further PAPI calls unless it again establishes itself as a PAPI agent (by calling InitPiAPI). TermPiAPI is always synchronous.

RESOURCE MANAGEMENT

AddPiResource()

AddPiResource() adds a server or class resource. AddPiResource is called by a PAPI agent to announce availability of a service, or to provide additional (class) information about a service that has previously been announced.

A given application program may announce one or more services, and may announce one or more classes for each service. The scope of availability of the resource may also be specified. This call may be synchronous or asynchronous.

RemovePiResource()

RemovePiResource is called by a PAPI agent to remove a previously added server or class. This call may be synchronous or asynchronous.

FindPiResource()

FindPiResource is called by a PAPI agent to locate a PIPES resource that matches the input parameters. If the pResourceInfo parameter is set to NULLP, the search is restricted to the local area. Otherwise, the pointer *pResourceInfo->fqPath* is used to globally identify the desired resource.

A resource is uniquely identified by the elements of *fqPath*, *resourceType*, *peerName*, and *className*. In the FindPiResource call, any of these may be *wildcarded* using the filename matching conventions of the shell for string parameters, and 0xff for resourceType.

The first match will be returned by this call; subsequent matches to the same input may be obtained by FindNextPiResource. This call may be synchronous or asynchronous.

FindNextPiResource()

After calling FindPiResource a PAPI agent may request another match to the same input by calling FindNextPiResource, and may continue until RESOURCE_ NOT_FOUND is returned. It is assumed that the calling parameters are identical to those of the most recent call to FindPiResource; results are undefined if this assumption is violated.

While not a requirement, calls to FindNextPiResource are meaningful only when some degree of wildcarding is present in the calling parameters, so that multiple matches are possible. This call may be synchronous or asynchronous.

SESSION MANAGEMENT

AllocPiSession()

This requests a conversation session with a server resource. AllocPiSession is called by a PAPI agent to request a session with a specific server resource. The PAPI agent, taking the role of a client requesting service, must either have prior knowledge of the server, or must locate it with calls to FindPiResource, FindNext PiResource. This call may be synchronous or asynchronous.

DeallocPiSession()

DeallocPiSession is called by either a client or server to terminate one or more active sessions. This call may be synchronous or asynchronous.

MESSAGE PASSING

SendPiMessage()

This routine is used by a peer to send a normal or expedited message to the other peer on the session. Normal messages are sent with guaranteed delivery and sequencing. Expedited messages are sent with guaranteed delivery, but no guarantee as to sequencing, either among themselves or in relation to regular messages.

They are treated with priority in the various stages of handling (such as queues), and in the assignment of system resources. This call may be synchronous or asynchronous.

RecvPiMessage()

RecvPiMessage is called by a peer to receive a message from the other peer on a session. More precisely, it enables the reception by providing resources and a destination for the message when it arrives. This call may be synchronous or asynchronous.

PROCESS CONTROL

SchedPiOS()

This schedules a notification when OS or user-defined services are available. An example of where this is useful is in those operating system environments, such as MS-DOS and Macintosh Finder, which are single-threaded for I/O. This call is always asynchronous.

SchedPiTime()

SchedPiTime is called by any PAPI agent simply to schedule a future notification at a specific time. The notification will not necessarily signify or be related to any external event. Input parameters passed to this routine are passed to the caller-supplied Notify routine, and can be used to distinguish among many scheduled notifications. This call is always asynchronous.

CancelPiRequest()

This allows previously made requests that have not been completed to be purged.

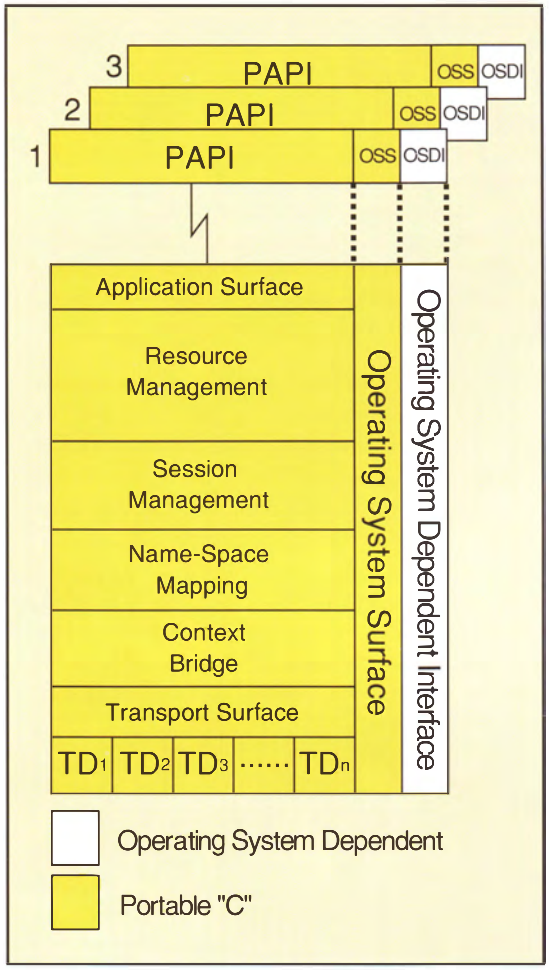


Figure 2. Layered Architecture of PIPES Platform

The Kernel Architecture

The PIPES kernel can be subdivided into a number of discrete subsystems, each handling a different aspect of the whole PIPES Logical Network (see Figure 2). The system is written in C and, apart from the System Dependent Layer and Transport modules, is portable to any host operating system. Migrating the PIPES Platform to a new environment involves the creation of a System Dependent layer and one or more Transport drivers, and then a compile and link in the new environment.

RESOURCE MANAGEMENT LAYER

The Resource Manager handles the creation and maintenance of the PIPES Logical Network by managing the addition, removal and replication of resource information throughout the network. The Resource Manager is also responsible for performing searches across the whole network for specified resources. The resource management services have the following characteristics:

Distributed and Replicated Name Space

Resource Information is distributed and replicated. In the PIPES system there is no central place where information is kept for retrieval. The information is stored at different locations that are transparent to the user.

Users do not require an understanding of the structure of the PIPES resource management scheme to retrieve information about the requested resources. In fact, frequently requested resource information is kept within the local machine, enabling faster access to these resources.

Consistent Name Space

Information about resources is constantly updated. Resource names can be dynamically added and removed. PIPES ensures that all copies of the resource information are updated within a finite time.

Location Transparent Resources

The physical location of the resources is hidden from the user. Resources can be moved from one location to another and PIPES will automatically update all of the resource information throughout the PIPES Logical Network. A PIPES application can specify the scope of availability of added resources. Information regarding these resources will not be transmitted beyond the specified scope. For example, access to a given resource can be restricted to areas within the local Group.

SESSION MANAGEMENT LAYER

The Session Layer is responsible for the establishment, maintenance and dismantling of sessions. Please note that users' messages are on a session specific to the user. The routing information and physical addresses are stored in the resource tables maintained by the kernel.

The Resource Management Layer communicates this information to the Session Layer while requesting a service from the Context Bridge. The Session Layer uses control packets to compute delays on different routes, link status and congestion. Based upon this information, the Session Layer automatically activates the flow control and congestion control. If a link goes down, alternate routes are searched (see Figure 3). If no alternate path is available, the session is terminated and the application is informed about the termination of the session. A session is always established with a resource. An application can establish multiple sessions with a resource.

CONTEXT BRIDGE LAYER

The PIPES Context Bridge Layer allows PIPES applications to communicate over heterogeneous and homogeneous communication networks. Once it is installed, every message received on a transport driver is checked for its destination. If the local session layer is the only agent that is supposed to receive this message, the message is simply passed on to the session layer.

However, if the message is intended for a different destination, the message is retransmitted on the appropriate driver. A message might have to traverse multiple Context Bridges before reaching its final destination.

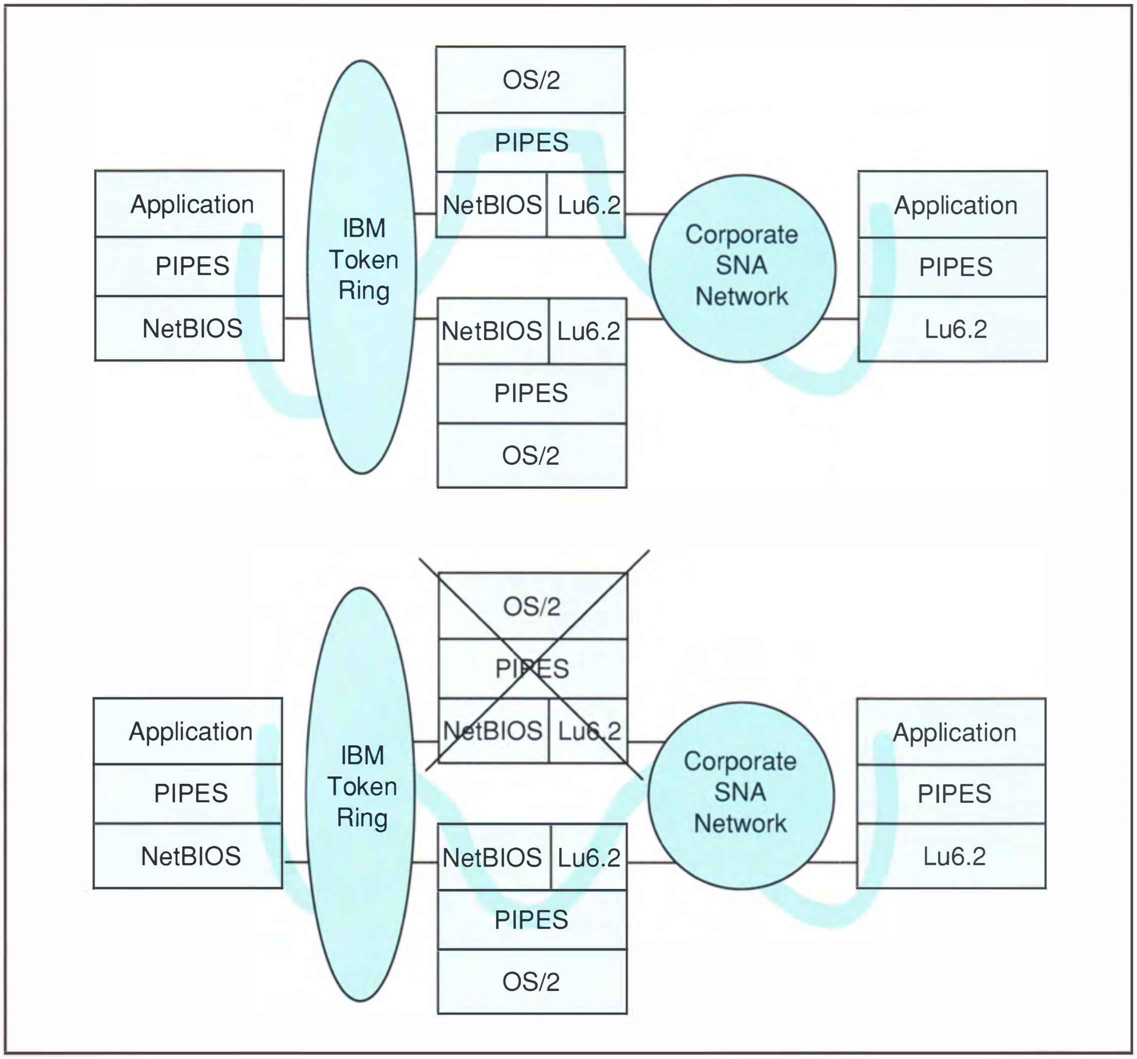


Figure 3. PIPES Platform's Automatic Recovery From Network Failure

TRANSPORT LAYER

The PIPES transport layer is composed of PIPES transport drivers. These drivers provide a consistent view of the underlying transport system to the PIPES Context Bridge Layer by using the services provided by the vendor-specific transport mechanisms. PIPES will be supporting all commonly used transport systems, such as NetBIOS, IPX, LU6.2, X.25, TCP/IP, and Appletalk.

The transport drivers provide a consistent interface to the PIPES Context Bridge Layer.

SYSTEM-DEPENDENT LAYER

The System Dependent Layer contains the PIPES code, which is specific to platforms supported by PIPES. This layer provides the following services to the PIPES environment:

- Timer Services
- Interrupt Management Services
- Memory Management Services
- Process Management Service
- Event Notification Services
- Data Transfer Services
- File Management Services

The System Dependent Layer must be rewritten for each new PIPES operating system environment, but contains all of the PIPES code which is dependent on that environment.

Peter Tait, *PeerLogic, 555 DeHaro Street, San Francisco, CA, 94107, (415) 626-4545. Mr. Tait is the Director of Product Planning at PeerLogic, the San Francisco developer of PIPES Platform. He is also a member of the InfoWorld Review Board and reviews networking products for that magazine. Mr. Tait has authored numerous articles, including two on distributed computing for LAN Times in December, 1990. He holds a BS degree in Physics from the University of Auckland.*

International Business Machines Corporation
P.O. Box 1328
Boca Raton, FL 33429-1328

G362-0001-09